PCT/AU2004/000839

Patent Office
Canberra

I, JULIE BILLINGSLEY, TEAM LEADER EXAMINATION SUPPORT AND SALES hereby certify that annexed is a true copy of the Provisional specification in connection with Application No. 2004902053 for a patent by NEOPRAXIS PTY LTD as filed on 16 April 2004.

WITNESS my hand this
Seventh day of July 2004

J. Billingsley

JULIE BILLINGSLEY
TEAM LEADER EXAMINATION
SUPPORT AND SALES

AUSTRALIA

Patents Act 1990

Neopraxis Pty Ltd

PROVISIONAL SPECIFICATION

Invention Title:

Movement Sensing and Monitoring System

The invention is described in the following statement:

Movement Sensing and Monitoring System

Field of the Invention

This invention relates to a system and method for monitoring and sensing the movement of an object or article, and more particularly relates to a system and method for sensing and monitoring the movement of humans in conjunction with the use of implanted circuitry to stimulate muscles and thereby control movement in a human. It also relates to a method and apparatus to enable stimulation of muscles of a subject in real-time such that the subject performs an activity.

Background to the Invention

In US Patent Application No 2001/0001125A1 there is disclosed a system for monitoring and/or affecting parameters of a patient's body consisting of an implanted system control unit (SCU) and implanted devices configured to be monitored or controlled by the system control unit through a wireless communications channel. The implanted devices include micro-stimulators, micro-sensors and micro-transponders. These are remotely programmed and interrogated via the wireless channel originating from external control devices such as a clinician programmer device or a patient control unit. The SCU can be programmed to transmit commands to the micro-stimulators according to a prescribed treatment regimen or periodically monitored biological parameters to determine a patient's status or the effectiveness of a treatment regimen.

There is a need to provide real time information on the movement of an object to enable a real time response for the purpose of adjusting or controlling subsequent movement of the particular object. For example where linear acceleration and angular velocity of a body part is measured in real time it is preferable to have an instantaneous response to control muscles, through an implant circuit, to perform various tasks such as sitting, standing or initiating a step for the object, and in particular a human, as a result of the measurement of acceleration and angular velocity. Alternatively such measurements of acceleration and velocity may be useful for further diagnosis and treatment.

The above mentioned document does not disclose a system that performs real time monitoring and measuring of the various parameters and more particularly does not disclose the measurement in real time of linear acceleration and angular velocity for calculating real time position and velocity of the moving object.

United States Patent No 5,919,149 discloses a system for monitoring postural sway of a human subject during standing or movement tasks. This is an aid in the

rehabilitation of balance and gait deficiency and more particularly provides feedback of postural information to the subject such as in a visual format by way of an imaging system mounted on a pair of eyewear used by the subject, or in auditory form or tactile form or alternatively in the form of electrical stimulation to the vestibular nerve.

In European Patent No 799597 there is disclosed an invention relating to a prosthetic apparatus that augments balance signals normally used by a human brain to control the body in a stable position and prevent a fall. It also provides information to a subject on its state of balance. More particularly though it relates to a balance prosthesis apparatus that measures angular velocity and position of a trunk of a subject in three orthogonal axes and provides information on the changes in the amplitude and frequency of these angular movements over time to the subject using either vibro-tactile, auditory, visual or direct electrical stimulation to the subject.

United States Patent No 6,063,046 discloses a method and apparatus that measures the balance response of a subject placed in a standing position on a support surface which is moved in any combination of pitch and roll directions. The measurements are made using force transducers, body sway sensors such as velocity transducers and EMG electrodes mounted over muscles on the left and right side of the subject's body. The measurements are displayed to an operator together with the response measurements from a normal sample population. From the displayed measurements the operator may diagnose the existence, cause and left or right side of a balance correction abnormality.

None of the above prior art documents disclose a system and method that measures linear acceleration and angular velocity of a subject to calculate the real time position and velocity of the moving subject over time with a view to altering or adjusting signals delivered to a prosthetic device in order to affect the movement of muscles of the subject to achieve a desired outcome such as walking, sitting or standing.

Summary of the Invention

According to a first aspect of the invention there is provided apparatus to enable stimulation of muscles of a subject in real-time such that the subject performs an activity, the apparatus comprising:

Sensor means for monitoring and measuring movement of the subject;

Controller means linked to the sensor means for receiving data from the sensor means indicative of the muscle activity; and

Stimulator means linked to the controller means for stimulating the muscles in accordance with commands delivered by the controller means in response to the data received from the sensor means.

According to a second aspect of the invention there is provided apparatus for detecting and recording movement of a portion of a subject in response to stimulation of muscles of the subject, the apparatus comprising:

Stimulator means for stimulating the muscles;

Sensor means linked to the stimulated muscles for recording data pertaining to movement of the portion of the subject;

Wherein the recorded data is forwarded to a controller means for analysis in real-time and whereupon a determination is made to send a command to the stimulator means based on the recorded movement data.

The portion of the subject may be any one or more of limbs of the subject, the torso of the subject, or muscles for controlling the bladder or seated pressure relief. The sensor means may include one or more accelerometers for measuring three dimensional linear acceleration of the portion and angular velocity measurements means, such as a gyroscope, for measuring the angular velocity of the portion. The acceleration and angular velocity of the portion is measured instantaneously and transmitted to the controller means.

The stimulator means may be in the form of a series of electrodes. The stimulator means may be a surface stimulator which is placed on the surface of the skin of the subject over the muscle to be stimulated.

Such a stimulator means may include electrode pads. The apparatus may be used to implement functional upright mobility, bladder control, seated pressure relief and lower extremity exercise of the subject. The sensor means may include any one of a torso sensor, a shank sensor, preferably one for each leg in the floor reaction orthosis(FRO) of the subject, a thigh sensor preferably one for each leg, a strain gauge, a posterior pressure sensor or an anterior pressure sensor, all preferably one for each leg. Preferably the pressure sensors are linked to respective shank sensors and preferably the shank sensor on each leg is connected respectively to each thigh sensor.

The controller means may be in the form of a body worn controller, worn externally on the subject, linked to each sensor means via a SPI bus.

According to a third aspect of the invention there is provided sensor apparatus for detecting movement of a portion of a subject comprising:

microcontroller means;

accelerometer means for measuring movement in at most three dimensions linked to the microcontroller means; and

angular velocity measurement means also linked to the microcontroller means for measuring angular velocity in one axis;

wherein the microcontroller means receives data signals from the accelerometer means and the angular velocity measurement means and after processing, forwards the data signals to a controller means for further analysis.

Each of the accelerometer means and the angular velocity measurement means may be linked to the microcontroller means through signal conditioning means. The sensor apparatus may further receive signals from a strain gauge and anterior and posterior pressure sensors. Preferably the signals are received by the microcontroller means through further signal conditioning means.

The sensor apparatus is preferably linked to the controller means through the microcontroller means via an SPI bus.

The angular velocity measurement means may be in the form of a gyroscope for measuring angular velocity of the portion of the subject in one axis of rotation and for providing angular velocity feedback to the microcontroller means.

The microcontroller means may have an analog to digital converter for converting analog data received from the signal conditioning means into digital data. The signal conditioning means that receives the output from the gyroscope may scale the output of the gyroscope so that the signal fills the available range of the analog to digital converter.

The accelerometer means preferably comprises two accelerometers, each of which measures acceleration in two of three dimensions but combine to measure all three dimensions. The signal from the strain gauge, which is preferably attached to the FRO, is preferably representative of the degree to which the subject leans into the FRO.

The sensor apparatus may further comprise a physical interface, preferably QPSI physical interface, that enables the controller means to send and receive data from the sensor apparatus, enable power transmission and programming from the controller means.

According to a fourth aspect of the invention there is provided controller apparatus for receiving data from sensor means that monitors and measures movement of a portion of a subject, the controller apparatus comprising:

a sensor interface means for sending commands to the sensor means and receiving data from the sensor means; and

transceiver means for sending power and commands to an implant device and for receiving telemetry data from the implant device.

The sensor means may be one or more sensors attachable to portions of the subject.

The controller apparatus may further comprise data storage means for storing data including means for identifying a sensor or multiple sensors. The apparatus may further comprise driver means, preferably in the form of a QPSI driver, linked to the sensor interface means for providing data communication to and from each sensor. The driver means may be in communication with a software module in the one or more sensors.

The controller apparatus may further comprise a strategy module linked to the sensor interface means for connecting, activating and/or deactivating the sensor means and for reading subject portion orientation from the sensor means.

The sensor interface means may store functions or programs that may be one of API public functions, non API public functions or private functions.

According to a fifth aspect of the invention there is provided a method of detecting and recording movement of a portion of a subject using computer program means comprising:

receiving analog values from one or more measurement devices;

converting the received analog values into digital values; and

storing the current digital values in memory means;

wherein controller means can access a specific current digital value from the memory means without waiting for a new analog to digital converted value.

According to a sixth aspect of the invention there is provided a method of communicating between a controller means and a sensor means using program means, such that command data is sent from the controller means to the sensor means, the method comprising the steps of:

on receipt of the command data by the sensor means, the program means undertaking the command according to the command data; and

transmitting a response to the controller means as a result of undertaking the command.

The command data may include any one or more of the following:

a). read digital values converted from analog values from measurement devices, the digital values being stored in memory means;

b) turning off measurement devices in the sensor means or external to the sensor means;

c) turning on measurement devices in the sensor means or external to the sensor means;

d) testing measurement devices to determine if an output results from the measurement devices;

e) calculating input physical values of acceleration, angle and angular velocity. The values of angle may be of the sensor means in the sagittal plane or coronal plane.

According to a seventh aspect of the invention there is provided a method of calibrating sensor means by program means to determine the orientation in space of the sensor means, the method comprising steps of:

placing the sensor means in a first position and measuring a number of consecutive values for all accelerometer and angular velocity measurement inputs;

determining the median of all values for each measurement obtained;

storing each determined median; and

repeating the above steps for each subsequent position of the sensor means.

The method may further comprise the step of searching through each of the medians of accelerometer values to determine a median corresponding to a predetermined minimum gravitational force and to determine a median corresponding to a predetermined maximum gravitational force. A zero level is then calculated as the average of these two values.

As all measured angular velocity values correspond to zero degrees per second, the most recent median value is used to set the angular velocity measurement (gyroscope) 0.

The method may further comprise the step of writing all values to a memory means in the sensor means. A controller means may transmit command data via a QSPI communications protocol to the sensor means to read the values stored in the memory means and to write to the memory means calibration constants.

According to an eighth aspect of the invention there is provided a communications protocol used by a sensor means and for communicating with a controller means, in the context of receiving data on movement of a portion of a subject, the protocol comprising:

a session layer for processing and forwarding data from an application layer and for sending to and retrieving data from a memory means;

a data link layer linked to the session layer for receiving and transmitting data between the session layer and a physical layer; and

the physical layer for undertaking communications between the controller means and the sensor means.

The physical layer may also communicate with transmit and receive buffers.

According to a ninth aspect of the invention there is provided a method of enabling communication between a sensor means and a controller means using a program, the method comprising the steps of:

using an application layer associated with the sensor means for receiving and processing commands from the controller means, performing the required command and responding to the controller means.

The method may further comprise the steps of executing a specific background task in the sensor means, when no other command process is being carried out by the sensor means, initialising the sensor means and calling any sensor means specific initialisation.

The method may further comprise the step of processing data to and from a data link layer and to and from a physical layer, through a receive buffer.

Brief Description of the Drawings

Preferred embodiments of the invention will hereinafter be described, by way of example only, with reference to the drawings wherein:

Figure 1 is a block diagram of apparatus depicting the invention according to one embodiment;

Figure 2 is a profile of a human having various sensors attached to the torso and legs together with a controller linked to each of the sensors;

Figure 3 is a block diagram of each of the sensors and gauges shown connected to the controller through a bus system;

Figure 3b is a block diagram of one of the sensors used in the invention;

Figure 4 is a circuit diagram of a 3.3 volt power supply;

Figure 5 is a circuit diagram of a 5 volt power supply;

Figure 6 is a circuit diagram of a 1.24 volt reference;

Figure 7 is a circuit diagram of a gyroscope and a gyroscope signal conditioning unit;

Figure 8 is a circuit diagram of an accelerometer that enables a self test function to be performed on the sensor;

Figure 9 is a circuit diagram of a signal conditioning unit;

Figure 10 is a circuit diagram of a strain gauge signal conditioning unit;

Figure 11 is a circuit diagram of a microcontroller of a sensor;

Figure 11a is a circuit diagram of the QSPI and programming interface;

Figure 11b is a circuit diagram of the microcontroller interface circuit that allows programming of the sensor;

Figure 12 is a circuit diagram for converting a measured strain into a voltage for on forwarding to a microcontroller;

Figure 13 is a circuit diagram of a QSPI physical interface;

Figure 14 is a circuit diagram of a circuit that enables the controller to read analog inputs via a sensor;

Figure 15 is a block diagram showing the architecture of the controller;

Figure 16 is a block diagram showing the connection of various sensors to the controller through a QSPI bus;

Figure 17 is a block diagram showing implementation of a sensor firmware including various protocol stack layers;

Figure 18 shows six different orientations that the sensors are placed in in order to calculate the certain values;

Figure 19 depicts angle measurement by one of the sensors;

Figure 20 shows a system for measuring the acceleration vector due to gravity using accelerometers;

Figure 21 is a diagram showing the architecture and implementation of a QSPI communications protocol;

Figure 22 shows a block diagram of protocol layers on the controller side and QSPI device side and the data flow therebetween;

Figure 23 is a block diagram illustrating various messages between each of the layers of the master controller and slave devices;

Figure 24 depicts a state diagram showing the general format of commands sent to a QSPI device;

Figure 25 is a state diagram regarding the QSPI identification command;

Figure 26 is a state diagram relating the session layer in a slave QSPI device;

Figure 27 is a state diagram describing the data link layer in a slave QSPI device; and

Figure 28 is a state diagram describing the operation of the data link layer in the controller.

Detailed Description of the Preferred Embodiments

The present invention is comprised of many components all of which have varying roles in providing subjects, such as humans, with renewed control over

muscles. This is particularly applicable to paraplegics where as a result of spinal chord injury they are no longer able to voluntarily control their muscles. In this particular embodiment benefits provided by the invention include functional upright mobility, bladder control, seated pressure relief and lower extremity exercise.

The various modes of operation available include a functional upright mobility mode which provides the patient with the ability to initiate movement of the body or to maintain stable body position. Specifically electrodes provide stimulation in a co-ordinated manner to implement the standing function , the step function once the patient is standing and the sit function also when the patient is standing. For all three modes closed loop control is achieved using various multiple sensor devices. A further mode is bladder control mode that provides the patient with the ability to initiate activation of the detrusor or sphincter muscles for the purpose of bladder control. Electrodes stimulate the patient's sacral roots to achieve bladder control. Another mode is the seated pressure relief mode which provides the patient with the ability to initiate activation of the gluteal muscles to adequately shift body weight and relieve seated pressure. Again the electrodes stimulate the gluteal and other muscles periodically to shift the patient's weight while sitting. A warning signal with delay shall be provided to the patient prior to initiating stimulation. A final mode is the lower extremity exercise mode which provides a patient with the ability to initiate movement of the lower extremities so as to increase muscle strength, decrease fatigue ability and increase muscle bulk. The muscles in the lower extremities shall be activated via electrode stimulation so as to produce an exercise effect, such as knee extension.

Shown in Figure 1 is a block diagram of the system with its various components that enables the monitoring and measurement in real time of the various parameters associated with the movement of a patient having an implant device. External to the patient there is a navigator controller 10, otherwise known as a Body Worn Controller (BWC) that drives the various sensor packs 12 and has a two way communication link with each of the sensor packs 12 to record the various measurements and also to enable data to be transmitted to a stimulator in response to the various measurements. The navigator controller 10 controls the operation of a surface stimulator 14 by sending to it commands over the interconnecting communication link which is a QSPI link. The surface stimulator 14 is a Body Worn Unit which includes a controller printed circuit board mounted in a housing with connections to the controller 10. The stimulation of the muscles is performed via electrode pads placed on the surface of the skin over the muscle. The sensor link or QSPI (Queued Serial Peripheral Interface) link provides power and communications between the sensor pack modules 12 and the navigator

controller 10 and more specifically provides power and control from the navigator controller 10 to the sensor modules and stimulator(s) and data from the sensor modules and stimulator(s) to the navigator controller 10. The sensors that make up the sensor pack 12 comprise body mounted sensor packs with connecting cables and connectors. They provide closed loop control in the upright mobility and exercise modes. In essence closed loop control means that commands transmitted by the navigator controller 10 are dependent, in part on the content of the data signals received by the navigator controller 10 as a result of feedback. The feedback from the sensors ensures safe stand up, sit down, standing and stepping functionality. They are self-calibrating and allow for drift in the sensors and movement of the skin. Power and control commands are sent from the navigator controller 10 to the implant device 18 via an RF link which includes a RF transmitting coil 16, a transcutaneous RF link that transmits the power and data to the implant across the skin of the patient which in turn is received by the implant device 18. The device 18 provides up to 22 channels of stimulation to a set of electrodes 20 that are used to transmit the stimulation from the implant to selected muscle groups. They are also used to transmit telemetry data back to the implant and in turn back to the navigator controller unit 10 over the RF link.

The implant 18 is a FES24-B which is similar in concept to that of standard and mini cochlear implants. It therefore has some common primary features with the cochlear implants, such as the RF antenna coil, coupling magnet and the receiver-stimulator and also has 22 electrodes connected with the extension leads and plug-in connectors together with bifurcated electrodes. Similar telemetry features available on the cochlear implants are also available on this particular implant 18. A remote control unit 22 enables patients to fully control live mode operation without necessitating direct interaction with the controller 10. Thus there will be a wireless communications link between the remote control unit 22 and the navigator controller 10.

Shown in Figure 2 is a profile of a human having various sensors attached to the torso and legs together with the controller 10 (Body Worn Controller). The sensors provide instantaneous three dimensional linear acceleration and angular velocity measurements which are fed back to the controller 10. These signals are then used to calculate the instantaneous position and velocity of the lower extremities of the human together with movement of the torso to enable control of the lower limbs during standing up, standing, sitting down and stepping operations. The system is designed to provide closed loop stable control of the lower limbs particularly of a paraplegic that is fitted with an electrical stimulation system such as the FES24.

In Figure 2 there is specifically shown a torso sensor 30, a shank sensor 32 one for each leg in the floor reaction orthosis (FRO) 36, a thigh sensor 34 again one for each leg, a strain gauge 38, a posterior pressure sensor 40 and an anterior pressure sensor 42. Each of the sensors are linked to the controller 10 with both pressure sensors 40 and 42 (on each foot) linked to a respective shank sensor 32. Each of the thigh sensors 34 and shank sensors 32 are strapped to the respective legs by suitable elastic means. Furthermore the shank sensor 32 and thigh sensor 34 on each leg are connected to each other. The strain gauge 38 along with the two pressure sensors 40, 42 are attached to the floor reaction orthosis 36 and are connected to the shank sensor 32 on the respective leg. Each sensor pack consists of up to two two-dimensional linear accelerators and one rate gyroscope which gyroscope measures the angular velocity.

The controller 10 has three sockets, into each of which a cable may be plugged. A number of sensor packs may communicate with the controller 10 along this cable, for instance two sensor packs on each leg will be linked to one cable. The chain of cables that can connect with a single socket is referred to as a channel. Each of the cables has eight pins terminated at the controller 10 with a modified RJ-45 plug. The cable contains connections for the SPI (Serial Peripheral Interface), power supply and an analog input for each branch. As mentioned previously there are three such cables connected to the controller 10.

Shown in Figure 3(a) is a block diagram of the various sensors and gauges connected to the controller 10 via the SPI bus 44. Shown in Figure 3(b) is a more detailed block diagram of one of the shank sensor packs 32. It includes a processing means in the form of a micro-controller 46 to which are linked a gyroscope 48 and a pair of accelerometers 50 and 52. Each of the gyroscope and accelerometers are linked to the micro-controller 46 through a signal conditioning unit 54, 56 and 58 respectively. Each of the strain gauge 38, posterior pressure sensor 40 and anterior pressure sensor 42 are linked to the micro-controller 46 via signal conditioning units 60, 62 and 64 respectively. The micro-controller 46 is linked via the SPI bus 44 to the controller 10. Separate power supplies and voltage references are also included in the sensor 32.

It is to be noted that each of the sensors will have a socket that allows analog connection to two additional pressure sensors and a strain gauge, and this is not limited to just the shank sensor 32.

The following description is with reference to the components of each of the sensors 30, 32 and 34. Each sensor has one or more power supplies for producing different voltages to specific components within the sensor. Generally the sensors have

a power supply voltage of between 3.4V and 5.25V. This is the input voltage range to the power supply which in turn produces an output regulated voltage of 3.3V in order to power the micro-controllers 46 and some signal conditioning units. It also produces a regulated 5V output supply in order to power the gyroscope 48 and some of the signal conditioning units. Finally a 1.24V reference voltage is produced from the power supply also for use in signal conditioning.

The 3.3V power supply (see Figure 4) also supplies power to the accelerometers 50, 52. It converts an unregulated voltage above 3.4V into a regulated voltage of 3.3V. The 3.3V power supply contains a National Semiconductor linear voltage regulator LP2951. The 5V power supply may provide power not only to the gyroscope 48 but also to the micro-controller 46 and accelerometers 50, 52. The supply may contain a Linear Technology Micro-power, regulated charge pump DC/DC converter LTC1516. The 5V power supply is shown in Figure 5. It is noted that the micro-controller pin PA0 (PIN51) is connected to the shutdown control of the 5V supply 80. This allows the micro-controller 46 to switch off the power supply for the gyroscope unit 48 and related signal conditioning unit 54. When the shutdown pin on the converter 80 is held low then the 5V supply is switched on. The combination of a ceramic capacitor 82 and tantalum capacitor 84 in parallel on the 5V output is used to minimise the switching noise. The micro-controller 46 has the ability to shut down the 5V power supply as mentioned previously by shutting or switching off power to the gyroscope 48 and gyroscope signal conditioning unit 54 which reduces the overall power consumption.

With regard to the 1.24V reference, the circuit is shown in Figure 6. Scaling of the output signals of the gyroscope unit 48 and the accelerometers 50 and 52 requires a voltage to be subtracted during signal conditioning. This entity is responsible for providing a stable 1.24V reference for use in this function. Thus the 1.24V reference is supplied to the signal conditioning units 54, 56 and 58 of the gyroscope and accelerometers. Again the input is an unregulated input voltage having a range of 3.4 to 5.25V and the output is a 1.24V signal having enough current capacity to supply the signal conditioning units 54, 56 and 58. As an example the 1.24V reference may contain a National Semiconductor Voltage Regulator Model No LM385.

The gyroscope unit 48 measures angular velocity in one axis of rotation and provides angular velocity feedback to the micro-controller 46 and in turn to the controller 10. The input supply is 5V and the output voltage represents angular velocity and a reference voltage which corresponds to zero angular velocity. Examples of gyroscopes that may be used are developed by Murata Model Nos ENC-05DD and ENC-05EE both of which are vibratory gyroscopes. These particular gyroscopes have

a scale of output of 1.11mV per °/s with a maximum of ± 90° per second. The voltage swing is generally about ± 100mV. The gyroscope unit 48 together with its signal conditioning unit 54 is shown in Figure 7.

With regard to the signal conditioning unit 54 it has the purpose of scaling the output of the gyroscope 48 so that the signal fills the available range of the analog to digital converter in the micro-controller 46. The input to the signal conditioning unit 54 is the voltage output of the gyroscope and the reference from the gyroscope. As can be seen from Figure 7 operational amplifier 90 has a positive input stemming from the output voltage pin of the gyroscope unit 48 and a negative input stemming from the voltage reference pin of the gyroscope unit 48. The reference voltage is generally smoothed and the difference between the voltage output and the voltage reference is filtered and amplified by the operational amplifier 90 yielding voltage V2. The 1.24V reference is then subtracted from the result by a second operational amplifier 92 so that the output lies within the available A/D input range on the input ADC0 pin (PF0) (PIN 61) of the micro-controller 46. The filtering is a combination of a low-pass filter at about 1Hz and a high-pass filter at 0.3Hz yielding a band pass filter structure as shown in Figure 7. The signal conditioning uses a National Semi-Conductor Dual Op Amp Model No RC LMC6462.

As mentioned previously the linear accelerometers 50 and 52 are two dimensional but combine to measure all three dimensions of acceleration. Each linear accelerometer measures two of the three dimensions of acceleration with one of the accelerometers being oriented perpendicular to the PCB so that the linear acceleration in the sagittal plane may be measured. Thus, one dimension may be measured twice to provide a reference and confirmation of the measurement. Also the particular direction of movement with respect to gravity can be determined. Each accelerometer produces two analog voltage outputs corresponding to two dimensions of linear acceleration. It also produces a pulse width modulated output corresponding to the two dimensions of linear acceleration. A self test facility is also included wherein the self test signal is taken from a digital output of the micro-controller 46. As an input to the accelerometer 3.3V is supplied from the micro-controller 46 and the various outputs are analog X – axis acceleration, analog Y – axis acceleration, PWM X – axis acceleration and PWM Y – axis acceleration. Specific suitable accelerometers that can be used are ADXL 202. A power cycling operation can be used by switching the power supply off in order to save power. The accelerometer 50 or 52 takes about 2ms to produce a stable signal after powering up. The bandwidth of the filtered analog outputs X_{FILT} and Y_{FILT} is 33Hz. The sensitivity of the analog outputs is 312mV per g. The PWM sensor outputs

X_{OUT} and Y_{OUT} encode the acceleration in the duty cycle of these digital outputs, where the duty cycle is defined as the ratio of T1 to T2 with T1 being the width of the pulse and T2 being the overall period of the on and off times in each cycle. The zero g reading is a 50% duty cycle and the sensitivity is on 12.5% per g.

With reference to Figure 8 a self test (ST) may be performed on the sensor by connecting the ST pin to a digital output of the microcontroller 46. The Vdd pins 13 and 14 are connected directly together and the COM pins 4 and 7 are connected directly together and these together with pin T2 are grounded. A decoupling capacitor C_{DC} is recommended to be connected between pin 14 and pin 4 having a value of approximately 0.1μF. The pins 9 and 10 for Y_{OUT} and X_{OUT} are connected to digital inputs of the microcontroller 46. The bandwidth of the analog outputs is set by capacitors C_x and C_y and calculated by the following formula:

$$F_{-3dB} = \frac{1}{(2\pi(32k\Omega) \times C_{(X,Y)})} = \frac{5\mu F}{C_{(X,Y)}}$$

The capacitors are approximately 150nF so that the 3dB cut off frequency for analog outputs is 33Hz. T2 is set using R_{set} using the following formula:

$$T2 = \frac{R_{set}\,(\Omega)}{125M\Omega}$$

Here R_{set} is chosen to be 1MΩ which yields T2 equal to 8ms.
The powerup time is given by the following formula:

$$T_{POWERUP} = (160C_{FILT} + 0.3)\ ms\ (C_{FILT} \text{ in } \mu F)$$

where its value is 24.3ms for analog outputs.

The signal conditioning units 56 and 58 for each of the accelerometers scale (amplify and filter) the analog outputs of the accelerometers so that the available resolution of the A/D channels of the micro-controller 46 are used. To fulfil this requirement there is 128 bits per g minimum resolution. Each analog input derived from the accelerometer will be amplified by a fixed factor and then a voltage subtracted to bias the centre point. An example of such a signal conditioning unit is manufactured

by National Semi-Conductor Model No LMC6462 Dual Op-Amp IC. An example of a circuit used for the signal conditioning unit is shown in Figure 9.

The strain gauge unit 38 and its corresponding signal conditioning unit 60 is shown in Figure 10. The strain gauge 38 is piezoelectric and produces an analog signal.

The signal conditioning unit converts the strain measured by the strain gauge into a voltage that is measured by the A/D converters in the micro-controller 46. The signal conditioning unit 60 amplifies the voltage signal from the strain gauge so that a measure of how much a subject is leaning into the FRO can be obtained. Strain gauge 38 maybe an AMP piezoelectric strain gauge with rivets DT1-028K and the strain gauge signal conditioning unit 60 may be a National Semiconductor Dual Op-Amp LMC6462. The output of the signal conditioning unit is connected to an A-D input of the micro-controller 46.

The micro-controller 46 enables the navigator or controller 10 to receive sensor readings via the SPI bus 44. It has several functions including allowing serial programming from the controller 10, converting analog signals from the sensor signal conditioning units into digital form, supplying power to the accelerometers, controlling the self test feature of the accelerometers and reading the digital outputs of the accelerometers. It also enables communication with the controller 10. An example of the micro-controller 46 used is the Atmel ATmega103L flash micro-controller. This has a flash memory, serial programming capability, eight channel 10 bit A/D conversion.

The microcontroller 46 is depicted in Figure 11. Pin Numbers 61 down to 54 represent pins that provide inputs to the analog to digital converter and are allocated pins PF0 (ADC0) to PF7 (ADC7). Respectively from pin 61 down to 54 they accept inputs from the accelerometer Y – axis signal conditioning unit, the accelerometer X – axis signal conditioning unit, the accelerometer Y – axis signal conditioning unit (perpendicular), the accelerometer X – axis signal conditioning unit (perpendicular), the strain gauge signal conditioning unit, the anterior pressure sensor 42, the posterior pressure sensor 40 and the gyroscope signal conditioning unit 54. Thus all of those signals are analog inputs. Digital inputs and outputs are passed through pins 51 down to 47, pin 35, and pins 6 to 9 inclusive. Pin 51 controls the 5V regulator so that the gyroscope 48 and accompanying signal conditioning unit 54 are powered down by asserting this HIGH. Pin 50 when asserted HIGH will self test the accelerometer A that is on the main board, pin 49 supplies power to this accelerometer, pin 48 when set HIGH self tests accelerometer B that is on the perpendicular board, and pin 47 supplies

power to accelerometer B. Pin 35 controls programming to the micro-controller 46, and pins 6 to 9 provide PWM signals to both accelerometers in each of the X and Y axis.

It is also noted that a low pass filter is inserted between the 3.3V supply and the A/D supply. The VCC pin is connected to the 3.3V and a 4MHz crystal 130 is connected between two pins as is a 32.768kHz watch crystal 132.

The signal conditioning units 60 and 62 have for their purpose converting resistance as measured by strain gauge 38 and pressure sensor 40 into a voltage to the input to the micro-controller 46. Shown in Figure 12 is an example of a circuit whereby a variable resistance measures the change in strain which is converted into a voltage and forwarded to the micro-controller 46. Each pressure sensor resistance change in the units 40, 42 has an input of 3.3V and an output voltage that represents the pressure exerted on the bottom of the FRO. The pressure sensor as an example may be a Flexiforce Single Serial Button or SSB-T force sensor.

As previously mentioned there are two accelerometers on each sensor pack. One of the accelerometers is oriented at right angles to the PCB so that all three dimensions of acceleration are measured. The accelerometer flat on the PCB shall measure acceleration along the PCB and transversely across the PCB. The other accelerometer shall be oriented so that one of its axes of measurement is perpendicular to the plane of the main PCB. The gyroscope has an output which is a scaled voltage that corresponds to the angular velocity of the PCB in the sagittal plane of the body that the sensor pack is mounted on.

A status light emitting diode is provided to indicate status information to a user of each of the sensors. The status LED enables status to be displayed to the user by allowing the micro-controller 46 to control the flashing of the LED. The LED will be constantly on if the micro-controller is in a programming mode or in a reset condition. The LED will flash under the control of the micro-controller in normal operation.

A QSPI and programming interface circuit allows the sensor 12 to be programmed via the same signal lines and connector as the SPI communications are transmitted on. The interface circuit connects either the SPI communication pins or the programming pins of the microcontroller 46 to the connector, depending on the output of the reset generation block signals. The interface is implemented using tristate buffers, diodes and resistors and the circuit is shown in Figure 11a. The interface controls the signals that appear at the QSPI_MISO (that is the master in slave out with the sensor being the slave) output of the connector. The MOSI (master out slave in) input is connected directly to both the RXD (programming input) and the MOSI pin on

the microcontroller 46. When the QSPI_RESET line is low, the QSPI_MISO signal will be equal to the TXD (programming output) pin of the microcontroller.

When the QSPI_RESET line is high, the microcontroller 46 is not in programming mode and communication may occur on the SPI bus. In this state, the microcontroller can effectively remove the sensor 12 from the bus 44 by setting PC1/TX_EN to high, which will make the sensor present a high impedance to the QSPI_MISO line. This is shown in the table below.

Inputs		Outputs
PC1/TX_EN	QSPI_RESET	QSPI_MISO
LOW	LOW	TXD
HIGH	LOW	TXD
LOW	HIGH	MISO
HIGH	HIGH	Hi-Z

The microcontroller 46 is programmed along separate pins to the SPI bus 44. The controller 10 has its efficient digital outputs to directly drive these pins. Thus a series of logic gates and tristate buffers are used to redirect the SPI bus interface to the serial programming interface pins. The microcontroller digital output PC0 is used as an IGNORE signal. When the pin is high, the SPI bus acts normally and when the pin is low, the microcontroller RESET signal is controlled by the chip select PCS from the controller 10. The logic table is shown below.

Inputs		***Outputs***					
PC0 (micro)	**PCS (BWC)**	**PDI (micro)**	**MISO (BWC)**	**MOSI (micro)**	**MISO (BWC)**	**RESET (micro)**	**SS (micro)**
L	L	MOSI (BWC)	PDO (micro)	Z	Z	L	L
L	H	Z	Z	MOSI (BWC)	MOSI (micro)	H	H
H	L	Z	Z	MOSI (BWC)	MOSI (micro)	H	L
H	H	Z	Z	MOSI (BWC)	MOSI (micro)	H	H

Shown in Figure 11b is the microcontroller interface circuit that enables the programming of the sensor. Before the microcontroller 46 is programmed the first time, the output at PC0 will be low. This will set IGNORE signal to low and hence the PCS is passed to RESET.

To program the microcontroller for the first time, PCS is held low to reset the microcontroller. PCS is held low while the microcontroller 46 is programmed.. When programming is completed, PCS is made high. The program in the microcontroller 46 will set the PC0 digital output to high on boot up. The serial communications is now connected to the SPI bus 44.

For reprogramming, once the software has been downloaded, a command is sent to the microcontroller 46 along the bus 44 instructing the microcontroller 46 to set PC0 to low. The PCS is then set low to reset the microcontroller while it is being programmed. There is the possibility of software errors making it impossible to reprogram the microcontroller 46. This can occur if PC0 is always high and cannot be made low. In this situation, there are two ways of programming the microcontroller. There will be some small jumpers connected to the RESET, PDI and PDO pins which can be used to reprogram the microcontroller. If reaching these pins is impossible, due to the sensor 12 being encased in Silastic, then pin 6 of J3 (strain gauge at pressure sensor connector) may be shorted to a GND pin, such as pin 5 on J3. This will set IGNORE to low and render the microcontroller 46 programmable. PC1 is used to ensure that if the sensor 12 is not supposed to be communicating then it presents a high impedence to the line.

Shown in Figure 13 is the QSPI physical interface that provides a means where the sensor can communicate, be programmed and receive power from the controller 10 through one of two connectors 140, 142. The interface provides a direct link between the power, ground and SPI signals and the connectors. The inputs are protected from electrostatic discharge whereby inductors L5 to L9 become resistive, at approximately 75 ohms at 100MHz. The communication lines are also protected from noise and voltages outside the voltage rails. The SPI communication signal inputs at lines 144, 146 and 148 are filtered with a low pass filter having a 3dB frequency of 1.6MHz. This helps filter high frequency noise from the signal line. Data is transmitted at a bit rate of 200kB/s. The SPI communication signal output on line 150 also has a low pass filter but a bandwidth ten times greater than that on the other output lines. This takes into account that several other sensors and other devices may be connected to the QSPI bus. In the event of above or below rail voltages being connected to the communication

lines, the BAV99 high speed diodes shown at 151 limit the voltage experienced by the sensor.

A reset generation circuit is connected between the micro-controller 46 and the controller 10 for the purpose of allowing programming of the micro-controller to be started by the controller 10 when there is software running on the particular sensor. The circuit puts the micro-controller 46 into a reset mode or programming mode when the PCS of the controller 10 is set low after the micro-controller 46 sets pin 35 to low.

External inputs are provided on the sensor 32 and the other sensors 30 and 34 that allow the controller 10 to read the analog inputs via the sensor. The sensor is forced into programming mode in the event of a software malfunction. The external inputs provide a direct connection between the analog inputs on the bottom connector of the sensor and the micro-controller 46. This block also provides protection against electrostatic discharge, noise and over-voltage conditions. The particular connector provides an external line that can force the sensor into a programming mode in the event of the above-mentioned software malfunction. The circuit is shown in Figure 14 which show the external analog inputs 160, 162 and 164 as including a low pass filter having a 3dB frequency of 1.6kHz in order to filter high frequency noise from the signal line. The external inputs can be pulled HIGH or LOW by the micro-controller by setting the signal on pin 27 HIGH or LOW. The micro-controller can also leave these outputs at a high impedance to not pull the inputs HIGH or LOW.

As mentioned previously the sensor pack 32 provides the ability for the controller 10 to control or read the sensor via the SPI bus 44. It also allows programming via the bus 44, calibration, and turning on and off components in the sensor. Furthermore it provides self tests of the accelerometers and reading of its analog inputs. Each sensor is able to calculate the dynamic orientation of the sensor in motion, measure angular velocity in one plane of motion, measure the acceleration components in three dimensions, measure the inputs from three general purpose analog devices such as from external sensors including pressure sensors and be able to control three digital outputs.

Each sensor enables the controller 10 to read a digital value that corresponds to the physical measurement of the following quantities:

angular velocity in the sagittal plane of the body segment that the sensor pack is in contact with;

linear acceleration experienced in three dimensions in relation to the body segment that the sensor is attached to;

sagittal angle of the body segment that the sensor is attached to;

coronal angle of the body segment that the sensor is attached to; and
external inputs as a voltage.

Controller Sensor Interface

Shown in Figure 15 is the overall architecture for the controller 10 which is in communication with one of the sensors 32. It is a master-slave arrangement with the controller 10 being the master and the sensor 32 being the slave. At the centre of the controller 10 is a sensor interface 170 which sends commands to a sensor pack application 172 and receives data from the sensor pack application 172. The sensor interface 170 is in communication with a strategy unit 174, a supervisor 176, a communications interface 178, a data storage unit 180 and QSPI driver 182. The driver 182 is in data communication with a software module 184 in the sensor 32. The driver 182 includes a QSPI port providing a hardware interface. The sensor interface 170 provides an application program interface or API to the strategy module 174 and the communications interface 178. The QSPI driver functions are called by the interface 170. All communications to the sensors are handled through the QSPI driver 182.

The sensor interface 170 provides a separate callable function for each sensor operation. It provides information about specific body segment orientation and angular velocity. Each relevant body segment is represented by an enumerated type. In a normal session the following operations are performed:

the sensor interface 170 is initialised by the supervisor 176;

the strategy module 174 connects and activates the various sensors; and

the strategy module 174 reads body segment orientation from any or all of the connected sensors and deactivates the sensors.

Thus the sensor interface 170 enables a strategy module to find the current orientation of specific limbs without needing to know specific sensor communication details. Each of the strategy modules call functions in the sensor interface 170 so as to obtain the current orientation of the body segments as measured by each of the individual sensors. Each of these functions run in the context of the calling task.

As mentioned previously a strategy module identifies specific sensors by an enumerated body segment type. Sensor interface 170 interrogates the data storage module 180 in order to find out the serial number of the sensor at this specific body segment. The serial number is used to address specific sensors via the QSPI driver 182 to send sensor commands to the specific sensor and receive sensor information from that sensor. The sensor interface 170 also maintains a register of a "replaced sensor". At initialisation, if one (and only one) expected sensor is missing and another sensor is

connected but not registered in data storage module 180, then the serial number of the unexpected sensor is stored along with the enumerated body segment of the sensor in the data storage module 180 that it is replacing.

The controller 10 preferably comprises an off the shelf pocket PC that is fitted with a custom designed CF+ interface card that has connections to the RF transmit coil 16 and a peripheral device link to the sensor packs 12 and the stimulator 14. The controller 10 uses the off the shelf pocket PC computer for handling controlling tasks and has a controller interface for interfacing to the pocket PC via the compact flash port of the pocket PC and provides functions as hereinafter described. The controller also includes a telemetry receiver means that is capable of receiving telemetry data transmitted by the CIC3 receiver stimulator within the implant 18. It also includes an RF transceiver means for transmitting the data to and from the controller 10 and a data encoder formatter that converts the stimulus information, that is active and indifferent electrode, amplitude, pulse width and interface gap, into the embedded protocol for transmission to the implant component. It also includes an RF interface to the remote control unit 22.

With regard to the software used in the controller 10, preferably the software under the name "Clinix" runs under the pocket PC's windows CE operating system. The software is preferably written in C++ and provides an environment for strategies for sit-stand-sit, stepping, exercise, bladder control and seated pressure relief to operate. Preferably the software's operation is configurable via a built-in programming system functionality and is designed so as to minimise the risk of unintentional stimulation. The software will not operate unless the controller interface is inserted into the pocket PC.

Functions that are contained in the sensor interface 170 are split into three different categories, API public functions, non-API public functions and private functions. The API public functions are those that will be supported for future versions of the sensor interface including future versions of the sensor hardware. Non-API public functions are those which are highly dependent on the current iteration of software and hardware and may not be supported in future versions of the sensors. Private functions are those that may only be called from within the sensor interface. Each of the sensor interface functions are mostly called by strategies however the communications interface 178 may call some sensor functions during a programming session to read sensor information and perform any required calibration. The supervisor 176 initialises the sensor interface. Each of the various functions are located in the file SensIntf.c. with the prototypes in file SensIntf.h.

The following is a description of the various sub-functions within each of the API public functions:

API Public Functions

Sensor Init

This function initialises the global data for sensor interface 170 so that there is no replacement device registered. The interface is initialised by the supervisor 176 by calling this function at start up. The function returns an enumerated error type **SensorErrorType**

SensorConnectMutiple

If this function is called by any strategy that requires sensor information it will be called when the strategy is started. The function connects all of the sensors, switches them on and checks them to see that they are working properly. The argument for this function is a pointer to an array of expected enumerated body segments for any expected sensors. There is also an argument for the number of expected sensors. The return type is an enumerated **SensorErrorType.**

SensorIsConnected

This function returns TRUE if the sensor is connected and FALSE otherwise. The argument is an enumerated body segment of a single sensor and the return type is a Boolean value.

SensorActivate

This function switches the designated sensor on. The function QSPISendPacket is used to send the command. If the sensor is turned on then SensorActiviate returns SENSOR_OK. If the sensor did not turn on, then SensorActivate returns SENSOR_ERROR.

The argument of this function is an enumerated body segment of a signal sensor and the return type is an enumerated type **SensorErrorType.**

SensorDeactivate

This function switches the sensors off. QSPISendPacket is used to send the commands and if the sensor is turned off and there are no communication errors, then this function returns SENSOR_OK. If the command was unsuccessful then the

function returns SENSOR_ERROR. The argument for this function is an enumerated body segment of a single sensor and the return type is an enumerated type **SensorErrorType.**

SensorVerify

This function verifies that a single sensor is operating normally. The QSPISendPacket is used to determine the status of the sensor and returns SENSOR_ERROR if there is an error that affects the functionality of the sensor. An enumerated body segment of a single sensor is the argument for this function and the return type is an enumerated type **SensorErrorType.**

SensorGetSagittalOrientation

This function returns the current value of the angle of orientation of the given sensor in the sagittal plane. If the angle was not obtained successfully then SENSOR_ERROR is returned. If the angle was obtained successfully then SENSOR_OK is returned. The arguments for this function is an enumerated body segment of a single sensor and also a pointer to the output value. The return type is an enumerated type.

SensorGetCoronalOrientation

The current value of the angle of orientation of the given sensor in the coronal plane is returned by this function. If the angle is not successfully obtained then SENSOR_ERROR is returned and if the angle is obtained successfully then SENSOR_OK is returned. The arguments are an enumerated body segment of a single sensor and also a pointer to the output value. The return type is an enumerated type.

SensorWriteCalibrationData

Calibration data in the input array is written to EEPROM of a specific sensor. The QSPISendPacket function is used to send this information. If the command is successful then SENSOR_OK is returned and if the command is unsuccessful then SENSOR_ERROR is returned.

The arguments include an enumerated body segment of a single sensor, a pointer to an array of **SensorCalibrationDataType** that contains the calibration data, and the length of the array of the calibration data is also included as an argument. This function has a return type that is enumerated.

SensorReadCalibrationData

This function reads all the calibration data of a specific sensor from its EEPROM into the return array. Again if the command is successful then the SENSOR_OK is returned and if not SENSOR_ERROR is returned.

The arguments include an enumerated body segment, a pointer to a returned array of **SensorCalibrationDataType** that contains the calibration data and lastly the length of the returned array of calibration data. The return type is an enumerated type. The calibration data shall be returned in an array of UINT16.

The following functions are considered **non-API public functions.**

SensorGetRawADCValues

This function allows the current value of the eight ADC values to be read from a specific sensor using QSPISendPacket. The argument is an enumerated body segment of a single sensor and this function returns an enumerated type **SensorErrorType.** The eight raw ADC values are returned in a variable parameter passed to the function as an argument. This variable parameter is a pointer to an array of eight UINT16 values.

SensorAccAXGetCalc

This function obtains the calibrated value of the acceleration measured in the X axis of accelerometer A, on a specific sensor. Again if the command is successful then the SENSOR_OK is returned and if not SENSOR_ERROR is returned.

The arguments include an enumerated body segment of a single sensor and INT16 which is a returned acceleration 2048 = 1g being 2048 bits representing a force of 1g. The function returns an enumerated type and the acceleration is returned in a variable parameter. The actual value of the acceleration is obtained by dividing the integer value returned by 2048 to give the value in g's. The returned acceleration is valid from –2g to +2g.

SensorAccAYGetCalc

This function obtains the calibrated value of the acceleration measured in the Y-axis of the accelerometer A on a specific sensor. If the command is successful then SENSOR_OK is returned and if not SENSOR_ERROR is returned.

The possible arguments are an enumerated body segment of a single sensor and INT16 which is a returned acceleration 2048 = 1g. This function has a return type which is enumerated. The acceleration is returned in a variable parameter. The actual

value of the acceleration is obtained by dividing the integer value returned by 2048 to give the value in g's. Again the returned acceleration is valid from -2g to +2g.

SensorAccBXGetCalc

This function obtains the calibrated value of the acceleration measured in the X-axis of accelerometer B on a specific sensor. As with the previous function if the command is successful then SENSOR_OK is returned and if not SENSOR_ERROR is returned. This function has the same arguments as the previous function with the same return type and derivation of the acceleration and range of the acceleration is the same as the previous function.

SensorAccBYGetCalc

This function obtains the calibrated value of the acceleration measured in the Y-axis of accelerometer B on a specific sensor. Similarly with reference to the previous function SENSOR_OK is returned if the command is successful. If the command is unsuccessful then SENSOR_ERROR is returned. It has the same types of arguments as the previous two functions and the same return type. Furthermore the derivation of the acceleration and the range in which the returned acceleration is valid is the same as the previous two functions.

SensorGyroGetCalc

This function obtains the current value of the angular velocity of a specific sensor and again if the command is successful then SENSOR_OK is returned and if not successful SENSOR_ERROR is returned.

The arguments include an enumerated body segment of a single sensor and a returned angular velocity 8 = 1° per second under code INT16. The return type is an enumerated type and the angular velocity is returned in a variable parameter. The actual value of the angular velocity is obtained by dividing the integer value returned by 8 to give degrees per second. The gyroscope returns values valid between +90 and -90° per second.

Finally there is one **private function** which is the **SensorGetSerialNumber** which returns the serial number for a given enumerated body segment. It returns the replacement sensor serial number if the enumerated body segment matches the replaced enumerated body segment. The argument is an enumerated body segment of a single sensor and the return value is a serial number of the sensor at the given enumerated

body segment. It returns a serial number of 0x00000000 if there is no device at that enumerated body segment.

The following description relates to software in the microcontroller 46 of any one of the sensor packs. The software is stored as code in a memory module, such as a EEPROM in the microcontroller 46. Particular aspects of the software include the slave application layer which includes the implementation of various commands, a calibration method used for the sensors, a device specific background task and a device specific initialisation.

As noted earlier, the primary function of the sensor pack is to measure angular orientation of the lower limbs and torso. The software in the sensor packs is divided into two portions, the communications protocol and the software that handles the sensor specific commands from the controller 10. The controller 10 sends sensor commands via the communications software to the sensor software where a command will then be performed and, if necessary, a response will be returned to the controller 10.

The primary function of the sensor pack will be implemented by software on the sensors themselves. The sensor packs keep track of their current orientation and the controller 10 then sends a command to read the current value of this information via the communications protocol.

Shown in Figure 16 is a block diagram of the various sensors linked to the controller 10 through SPI buses 44. Specifically the torso sensor pack 30 is directly linked to the controller 10. Each of the left thigh sensor 34a and left shank sensor 32a is linked on the one SPI bus to the controller 10 as is the right thigh sensor pack 34b and right shank sensor pack 32b. Each of the sensor packs are regarded as SPI slaves and the controller 10 is regarded as a SPI master.

In Figure 17 there is shown an implementation of the sensor pack firmware which includes various protocol stack layers and various modules to be described hereinafter.

The analog input sampling process 200 constantly samples the analog inputs corresponding to four accelerometer outputs, that is AX, AY, BX and BY and also one gyroscope output, one strain gauge output and two pressure sensor outputs. The current values are then copied into memory. When the controller 10 requests a specific value the memory location is read and transmitted back without having to wait for a fresh analog to digital conversion. It enables an increase in speed at which the controller 10 can read specific values from each of the sensors. The process effectively is an interrupt handler. The analog to digital conversion must be initialised by calling the **InitAnalogInputSampling()** function which is called as part of the QSPI device

specific initialisation in the function InitDevice() at reset. The most recent analog output values will be located in RAM. The high byte and low bytes are located in separate arrays. Interrupts must be disabled when reading from these values to prevent partially updated values from being read.

A further function that is used which is also an interrupt handler for the ADC Conversion Complete Interrupt is the function **AnalogInputSamplingISR**. This function copies the current value to memory, changes the ADC channel register (ADMUX) to the next channel to be sampled. Two 8 byte arrays are used as buffers for the high and low bytes of the ADC channels. One array is used for the high byte and another array is used for the low byte. Each member of the array corresponds to one of the eight ADC channels. A simple semaphore is implemented in this interrupt handler by disabling interrupts while the global memory is being written to.

Finally the function **GetAnalogInputSample** allows reading of the latest value of an ADC channel automatically. Interrupts are disabled while both the ADC bytes are being read.

The slave application layer 202 in Figure 17 is the micro-controller code that receives commands from the controller 10, carries out the required action, then transmits the appropriate response. The functions of this entity execute in the context of the main thread 204.

The function of the application layer 202 is to receive commands from the controller 10, perform the desired action and respond. The following functions are performed:

Analog input sampling where the buffered values written by the ADC interrupt are read;

turning off the sensors such that upon receiving the appropriate command from the controller 10, the sensor components such as accelerometers and gyroscope will be turned off;

turning on the sensors, wherein upon receiving the appropriate command from the controller 10, the sensor components such as accelerometers and gyroscopes will be turned on;

self testing of sensors, whereupon receiving the appropriate command from the controller 10, the sensor components such as accelerometers and gyroscope will be tested to see if they are producing an output; and

calculations on input values wherein the calibrated physical values of acceleration, angle and angular velocity are calculated.

A particular message **sesPcktArrive** is input to the slave application layer 202. This message is processed by the function ProcessDevicePacket called by the session layer. Other messages which are input to the slave application layer being **appDATA, appCMD** and **appDisconnect** are processed as part of the first mentioned message **sesPcktArrive** and hence within the function ProcessDevicePacket.

All of the commands and messages from the master application layer are handled via the function ProcessDevicePacket, which decodes the packet and takes the appropriate action and then sends the correct response/data. The function first looks at packetData[0]. The most significant nibble indicates the network address and the least significant nibble indicates the packet type. If the network address is equivalent to the current network address of the sensor then the packet processing continues. If the packet type is a sensor command (appCMD) then the following byte indicates which command is being sent. If the packet type is a general data packet (appDATA) then a simple response packet is sent back. There are no commands configured to use this packet type with sensors. Any other packet type is not processed by the ProcessDevicePacket function.

If a sensor command has been sent, the function looks at packetData [1] and compares it to the enumerated type SensorCommandType. Then the appropriate command is executed and an appropriate response packet is returned using the sesPcktSend function in the SPI session layer. When responding, a different packet type appRESP (sensor response) is used.

The ADC module contains an interrupt handler that copies the analog input registers to a memory array after each conversion, as mentioned previously. The application layer module 202 accesses this array via an **extern** declaration. The GetAnalogInputSample function is not used to return the raw ADC values as this is less efficient than sending the bytes separately without combining and splitting a 16 bit value first. Two different byte arrays of eight bytes each are provided, with one byte containing the high byte from the analog to digital conversion register (ADCH) and the other containing the low byte from the analog to digital conversion register (ADCL).

The parameters are stored in the memory of the micro-controller 46, typically a EEPROM, and are accessed via the session layer. The shaded EEPROM data is present on all QSPI devices. The parameters include analog X and Y values derived for each of the accelerometers A,B at quick calibrate values -1g, 0g and +1g, providing a total of twelve parameters. It also includes the parameter of the gyroscope 0deg/sec value. Commands from the master application layer to the sensor slave application layer correspond to the parameters such that they control the reading and writing of the

parameters stored in memory. The values stored in memory are loaded to RAM at start-up and when the UPDATE_PARAMS command is received. The read and write commands that read and write the values from the memory EEPROM returns them to the controller unit 10. A typical command may be RD_QCAL_AXL for example that reads the quick calibrate value –1g for the accelerometer Analog X value, would trigger the following sequence of operations:

1. RD_QCAL_AXL received;
2. Read the two bytes from the appropriate EEPROM address, the high byte being read first.
3. Send back an appRESP to the command.

The general EEPROM commands allow reading and writing of any byte in the 4K of on board EEPROM by the controller unit 10. Hence the controller 10 can update the EEPROM directly. The actual parameters in memory will not be updated until the UPDATE_PARAMS command is sent from the controller 10. Most EEPROM commands are implemented in the application layer. QSPI Identification commands access the serial number, QSPI Device Number and Software Version Number which are implemented in the session layer of the communications software.

The following description relates to the calculated values which are performed by the sensors and include angular orientation, angular velocity and acceleration in three dimensions. The calculations are performed in the context of the main thread 204 in a user-defined Background Task 203. The Background Task 203 is a generic function called by QSPI Peripheral Device Communications software as part of a main loop that handles the communications processing. This main loop and the Background Task 203 may be interrupted at any time by any system interrupts while the respective interrupt handlers execute. The purpose of the sensors is to measure the orientation of the sensors in space. The sensors have the capability of calculating their own angular orientation and the calibrated values from the gyroscope and the accelerometers can be calculated. The following describes the method of calibration and the use of the calibration coefficients to calculate the real values of the various parameters listed below:

Accelerometer A X-axis Acceleration;
Accelerometer A Y-axis Acceleration;
Accelerometer B X-axis Acceleration;
Accelerometer B Y-axis Acceleration;
Gyroscope Angular Velocity;
Angle of the sensor measured in the sagittal plane; and

Angle of the sensor measured in the coronal plane.

The angular orientation calculation is carried out in the function CalculatePosition. This function is called by the DoBackgroundTask function which in turn is called in the main thread 204. Other calculated values are calculated by the following functions:

ExecuteCMD_RD_AXcalc
ExecuteCMD_RD_AYcalc
ExecuteCMD_RD_BXcalc
ExecuteCMD_RD_BYcalc
ExecuteCMD_RD_GYROcalc
ExecuteCMD_RD_ALLcalc
ExecuteCMD_RD_ANGLEcalc
ExecuteCMD_RD_CORONAL_ANGLE

The above eight functions are called from the function ProcessDevicePacket.

The calculated angle, acceleration and gyroscope values assume that the sensors have been calibrated in the following way. The sensors can be oriented in six different ways, once for each of the X, Y and Z axes and another three ways corresponding to the opposite orientation of these axes. The sensors are placed in each of these six positions which exposes each of the sensors to +1g and –1g on each of the three axes. The six different orientations are shown in Figure 18.

The calibration algorithm is implemented externally to the sensor pack 32, for example. The controller 10 uses the SPI bus 44 to command the sensor 32, read the data and write the appropriate calibration constants to the memory unit in the micro-controller 46. The calibration of the accelerometers 50 and 52 are carried out as follows:

1. The sensor 32 is placed in a first position, held still, then 32 consecutive values are measured for all of the accelerometer and gyroscope ADC inputs.
2. The median of all the values for each ADC measurement is found. A noise resistant estimation of the calibration value for all of the accelerometer and gyroscope ADC inputs for this position is determined.
3. The medians in a row of an array are stored.
4. Steps 1 to 3 are repeated for each of the six positions, creating six rows.
5. Each column of the array of medians of the accelerometer values is searched for each of AX, AY, BX and BY. The minimum median measured for each corresponds to the –1g level. The maximum corresponds to the +1g level with the 0 level being calculated as the average of these two values.

6. All measured gyroscope values correspond to 0° per second. The most recent median value is used to set the gyroscope zero point. This sets the zero point but not the gain as the gain is not set in this procedure.

7. All of the values are then written to the memory EEPROM in the micro-controller 46 in the sensor 32 using commands previous mentioned.

8. The sensor 32 is now calibrated.

The accelerometers 50 and 52 are able to detect the acceleration due to gravity. This factor is used to calibrate each of the sensors. When the sensors are placed on a flat surface, the acceleration experienced by each axis of the accelerometers will be 0g if the axis is horizontal with respect to ground. If the axis being measured is vertical with respect to the ground then the acceleration experienced will be +1g or –1g depending on the orientation of the sensor. Hence by placing the sensor 32 on a flat surface, measuring the accelerations and then placing the sensor upside down, one axis of the accelerometers may be calibrated by considering the values equal to +1g and –1g respectively. All axes may be calibrated in this manner. The Sensor Interface Test Harness used for calibration requires that the sensor 32 be placed in the six possible positions for calibration of all three axes of acceleration. The order in which the sensor pack 32 is placed in these positions does not matter for the calibration algorithm used. As mentioned previously the six different positions are represented in Figure 18.

The acceleration measured by each of the accelerometers 50 and 52 is measured as a voltage by the micro-controller ADC. The micro-controller 46 is constantly sampling the inputs, leaving the current voltage measurement in memory. The voltage measured is represented as a ten bit unsigned integer stored as two unsigned 8 bit integers. The following formula is used to calculate the acceleration measured:

$$a = \frac{k(A_{raw} - A_0)}{A_H - A_L} \qquad (1)$$

where A_{raw} = the 10 bit raw ADC value for one of AX, AY, BX, BY.

A_0 = the 10 bit calibration 0g value of AX, AY, BX, BY.

A_H = the 10 bit calibration +1g value of AX, AY, BX, BY.

A_L = the 10 bit calibration –1g value of AX, AY, BX, BY.

k = scale factor. The output is a signed 16-bit integer where a value of k/2 corresponds to 1g.

a = acceleration, in g's multiplied by k/2.

The acceleration is returned as a signed 16 bit integer in 2 bytes, using k = 4096 so that 2048 is equivalent to 1g and –2048 is equivalent to –1g. The form of the equation (1) is chosen to minimise the quantisation and round off errors due to integer arithmetic.

Regarding the calculation of angular velocity, a nominal gain for the gyroscope signal conditioning circuit of 6.25 is assumed. Therefore ± 90° per second responds to ± 0.625V around the zero angular velocity voltage. The value 3.3V corresponds to an analog to digital converted value of 1024, with 3.223mV per division and a 10 bit resolution.

The angular velocity is represented by the following equation:

$$\frac{\omega = k(G_{RAW} - G_0)}{k_g} \quad \text{deg.}s^{-1} \qquad (2)$$

where G_{RAW} = ADC gyro value.

G_0 = Gyro zero value estimated using a median filter

k = 8192

k_g = a value calculated using a previous method

This yields a signed integer scaled so that a value of 8 corresponds to 1° per second.

If ω = 8, the angular velocity is +1°/s

Similarly, if ω = -8, the angular velocity is –1°/s

The parameters k and k_g have been chosen to ensure that using integer maths to calculate equation (2) minimises errors due to rounding and quantisation.

The sensor 32 combines information from both the accelerometers 50 and 52 and the gyroscope 48 to calculate the current angular orientation of the sensor with respect to gravity. A signal processing technique called "Wiener Filtering" is used to combine the accelerometer values and the gyroscope values to get a better estimate of the sagittal angle than by using either estimate alone. Reference is made to the text "Introduction to Random Signals and Applied Kalman Filtering" by R G Brown and PYC Hwang, 3rd edition John Wiley & Sons, ISBN 0471128392. This technique utilises the fact that the gyroscope generated angle (integrated angular velocity) is good for dynamic situations and the accelerator generated angle is good for static situations. The sensor is designed to measure its angle with respect to the horizontal plane along

its axis. Angle measurement by the sensor 32 is shown in Figure 19. Further information on Wiener Filtering is provided in Appendix 1

The accelerometers 50 and 52 can be used to measure the acceleration vector due to gravity as is shown in Figure 20. In practice, the sagittal plane is represented by the plane defined by the Z and X axes. These three components of the gravity acceleration vector are available from the calculated acceleration values from the accelerometer A or B as shown in Table 1 below.

Accelerometer	ADXL202 Axis	Variable Measured
A	X	Ax
A	Y	-Ay
B	X	Az
B	Y	-Ax

The angular orientation of the sensors can be completely described using three degrees of angular freedom. However, only two degrees of freedom may be obtained from the acceleration value detected due to gravity. The angular orientation that corresponds to a "compass bearing" cannot be determined from the gravity acceleration vector. The two angles that can be obtained from the gravity acceleration vector are referred to as the sagittal and coronal angles. These angles measure rotation in these anatomical planes.

If the sagittal angle θ and coronal angle φ are known and the sensor pack is still, the acceleration measured by the sensor pack is given as:

$$a_x = g \cos(\varphi) \sin(\theta)$$
$$a_y = g \sin(\varphi) \qquad (3)$$
$$a_z = g \cos(\varphi) \cos(\theta)$$

In practice the sagittal and coronal angles are calculated from the measured values of a_x, a_y and a_z. The parameter g is the magnitude of acceleration due to gravity. The coronal angle φ is calculated on a continual basis using the measured a_y value where the absolute value of a_y/g is limited to one so that the inverse sine function returns a valid answer. The coronal angle lies in the range of $-90° < \varphi < 90°$.

The sensor angle or sagittal orientation θ may be calculated from either the angular velocity measurement previously referred to or the acceleration measurements also previously referred to. The angular velocity measurement can be integrated over time to obtain an estimate of the sagittal angle. The detected acceleration can be used

to calculate the angle by assuming that the acceleration is due only to gravity and by using trigonometry to calculate the sagittal angle θ.

Both of these measurements are not always correct. When the sensors are moving, the total acceleration measure consists of two components, being gravity and acceleration due to the motion of the sensor, that is:

$$\underline{a_t} = \underline{g} + \underline{a_m}$$

where a_t is the total acceleration vector measured, a_m is acceleration vector due to motion of the sensor and g is the acceleration vector due to gravity.

Hence the angle derived from the acceleration values contains errors due to the movement of the sensors. The angular velocity measurement and its derived angle is substantially accurate when the sensors are moving. When the sensors are still, the angle derived from the accelerometers is reliable. However, the integrated angle derived from the angular velocity measurement is susceptible to accumulating error due to small offsets in the calibrated zero for the gyroscope.

The angle θ shown in Figure 19 is derived from the acceleration values by using a 4- quadrant inverse tangent function. The 4 quadrant version of the inverse tangent function is used so that the full angular range of –180° to +180° can be derived from the measured accelerations using the equations given for a_x, a_y and a_z in (3) above.

The angle θ in Figure 19 is derived from the angular velocity values by integration. A trapezoid method approximation of integration is used to do the numerical integration. This method has a particular advantage of allowing variable time steps which is useful as the angle calculations are running in the background with variable timing.

The particular equations are:

$$\theta_n = \theta_{n-1} + \int_{tn-1}^{tn} \omega \square dt$$

$$= \theta_{n-1} + \frac{(\omega_n + \omega_{n-1})}{2} (t_n - t_{n-1})$$

The sagittal angle value θ returned by the sensors is a signed 16 bit integer scaled so that 128 corresponds to 1°. The return value is –250° to +250°.

In appendix 1 there is a description of Wiener filtering in relation to estimating the sagittal angle from the two angle estimation sources above mentioned.

With regard to the implementation of the filters there are two first order filters in the estimator (referred to in Appendix 1) that must be efficiently implemented in the

sensor micro-controller 46. The filters are implemented as discrete filters assuming a sampling rate of 128 Hz which is the average analysis rate in the implemented system. The filters are transformed to the discrete time domain using the bilinear transformation:

$$s = 2F_s \frac{1 - z^{-1}}{1 + z^{-1}} \qquad (4)$$

where F_s is the sampling rate.

The transfer function is converted to difference equations as follows:

$$Y(s) = \frac{1}{\tau s + 1} X(s) \Rightarrow y_i = \left[\frac{2F_s\tau - 1}{2F_s\tau + 1}\right] y_{i-1} + \frac{x_i + x_{i-1}}{2F_s\tau + 1} \qquad (5)$$

The filters are implemented using integer multiplications and divisions with the input being multiplied by 16. The value of the output of each filter is 16 times the actual filter value and this reduces the effects of excessive quantisation in the calculation of the filter outputs.

The sagittal angle estimation algorithm must be initialised soon after the sensor is activated by turning on the gyroscope and the accelerometers. The function that turns on the gyroscope calls the InitCalculatePosition function when the gyroscope has finished initialisation. The InitCalculatePosition function initialises the integrator and filter so that the output of the sagittal estimation algorithm is an angle derived from the accelerometers.

As mentioned previously a built-in test runs continually when the sensor 32 is activated. The built-in test causes the LED to flash quickly when the test fails, informing the users that the sensors should not be used. The test also fails under the following conditions:

1. If the long term average, calculated by the pseudo-median filter described in Appendix 2, of the gyroscope ADC value falls outside a certain range of the calibrated 0°/second value (within 50 ADC levels);
2. If the redundant axes of the two accelerometers 50 and 52 do not match for a significant period of time, that is outside 0.25g difference for 400ms; and
3. If the gyroscope signal is stuck at a constant voltage.

This last condition is a known failure mode of the gyroscope when subjected to shocks exceeding its maximum rated shock. This failure mode is detected when the

sagittal angle estimation derivative is non-zero for a period of time, typically greater than 400ms, while the gyroscope signal is at a constant value.

The built-in test conditions also fail when the sensor is not calibrated or there is a serious problem with the accelerometers or gyroscopes. When a calculated value is read by the controller 10 and the built-in test failure could affect the value to be read, then a built-in test error byte is returned along with the current value.

Details on the pseudo median filter used are in Appendix 2 and further details on the gyroscope calibration program is given in Appendix 3.

The following is a description of the design of the micro-controller software on any QSPI device. Furthermore the communications protocol used by the micro-controller 46 is also described. Non-device specific firmware in any QSPI device is also described hereinafter and includes slave QSPI protocol including session layer, data link layer and physical layer and also the slave application layer shell.

Referring back to Figure 16 each of the devices 32a and 32b being the left and right shank sensors, devices 34a and 34b being the left thigh and right thigh sensors and device 30 being the torso sensor are considered to be QSPI devices and SPI slaves. The controller 10 is considered to be a SPI master. The software in a QSPI device allows the controller 10 to communicate with a number of QSPI devices using the QSPI device and communications protocol. An application in the controller 10 sends commands to each device application using this protocol and the slave device performs this command and returns data if appropriate.

Shown in Figure 21 is a diagram similar to Figure 17 that includes a session layer function 205, a data link layer function 207 and a physical layer function 209. The physical layer function 209 is a slave layer that performs any operations involved for the SPI bus 44 in relation to the micro-controller 46. This layer handles the circular buffer for transmitted or TX and received or RX data. The operations are executed in the SPI transmission complete interrupt handler and the main thread 204.

The main thread 204 is the entry point for the initialisation, QSPI Device Specific Background task and for processing communications. The initialisation is divided into SPI initialisation and QSPI device specific initialisation. The latter initialisation function is provided by the QSPI Device Specific firmware. The second task being the QSPI Device Specific Background task is executed when no communication processing has been carried out. This function is supplied in the QSPI Device Specific firmware. The processing of received bytes from the physical layer buffer is carried out in the main thread 204.

The data link layer function 207 is also a slave layer and receives and transmits variable length packets with error detection. This entity executes in the context of the main thread 204 and the operations of this layer are described more fully later in the context of QSPI communication protocols.

The session layer function 205 is also a slave layer that controls link configuration and management and acts as a conduit for the application layer 202 commands and responses. This entity executes in the context of the main thread 204 and the operations of this layer are described more fully in relation to the QSPI communication protocols later on. The session layer 205 also handles QSPI Device Identification commands and some other non-device specific functions.

An application layer 202 is also a slave layer function which is the micro-controller code that receives commands from the controller 10, carries out the required action and then transmits the appropriate response.

In terms of file structure the code for any QSPI device is divided into two parts, being device dependent and device independent.

The main thread function 204 has for its purpose processing communications and executing the device specific background task. The functions of the main thread 204 are to perform global initialisation of the processor, call any device specific initialisation, execute a QSPI device specific background task and process communications. It assumes that the functions InitDevice and DoBackgroundTask are defined in the QSPI device specific code. The initialisation values are written to their respective control registers at start up and on reset . At reset, the processor is initialised using the InitialiseProcessor function to be described hereinafter. The main thread 204 consists of an endless loop that handles communications processing and the device specific background task.

The InitialiseProcessor function has no arguments or a return value type as is the case with the main thread function or entity. The InitialiseProcess function performs global initialisation to enable global interrupts, set the RAM page, disable clock division, set the processor to be not sleeping and make sure there is no external SRAM. The physical layer, data link layer and session layer initialisation functions are called from this function directly. QSPI Device specific initialisation is carried out in this function by calling InitDevice(), defined elsewhere in device specific code.

The main thread 202 as mentioned previously consists of an endless loop that calls a particular function in the data link layer 207 function when there are unprocessed bytes in the receive buffer. If there are no bytes in the receive buffer to be processed then the device specific background task function will be called.

The slave physical layer 209 is a communications protocol layer and its function is to allow basic communication with the controller 10 using the SPI of the micro-controller 46. The physical layer also provides an interface between the micro-controller SPI and the higher communication protocol layers. The operation of the physical layer assumes that the layer has been initialised using the InitPhysicalLayer function defined in the Functions section of this entity. The physical layer is implemented on the micro-controller 46 configured to be in slave mode communicating on the SPI bus.

The physical layer has two situations where processing must be carried out, that is sending bytes and receiving bytes. The layer provides a software implemented circular buffer on input and output data from the SPI. The SPI Complete interrupt handler attends to the receiving of bytes, storing the bytes in the RX circular buffer and transmitting bytes from the TX circular buffer. The processing of the bytes occurs in the main thread as previously mentioned.

When a new byte arrives in the SPI data register or SPDR, the SPI interrupt is raised. The SPI interrupt handler reads the incoming bytes into the physical layer circular buffer. When the interrupt handler has finished execution the main loop or thread 204 resumes and the bytes in the circular buffer that haven't been processed are then processed by a called particular function. The function call corresponds to a message from the physical layer to the data link layer indicating that an SPI transmission has just occurred. When the SPI interrupt handler is called and there is a byte in the TX circular buffer waiting to be transmitted, then the next TX buffer byte is written to the SPDR after the SPDR is read.

Three functions defined in the Functions section in particular are used in the slave physical layer being InitPhysicalLayer, dlSendAndReceiveByte which writes a byte to the circular buffer for sending on the SPI bus 44 and returns the last byte received directly from the SPDR. Furthermore there is the function SPI_ SerialTransferCompleteISR which is an interrupt handler triggered every time the byte arrives on the SPI bus.

The slave data link layer 207 is a communications protocol layer that sends and receives packets between the controller or master 10 and the micro-controller 46 which is the slave. Error detection is also carried out. The layer 207 receives messages from the three sources, the physical layer, the session layer and the master data link layer. These messages are implemented as C functions that manipulate the data of the slave data link layer. The functions that are used as an interface to the DLL include plTxFinish, sesPcktSend and InitDataLinkLayer.

The function sesPcktSend acts as a message to the DLL from the session layer. The function turns the input data into a packet by adding a start byte, a length byte and a checksum. The packet is also byte stuffed which involves the process of replacing each start byte in the message by two consecutive start bytes. The function corresponds to a message from the session layer to the data link layer instructing the DLL to send a packet to the master device. As all communications with the controller 10 are controlled by the controller 10, this means that all the bytes of the packet are written to the SPI data register by the physical layer until they are all read by the controller 10.

If the current DLL state is IDLE then the following actions take place, the MISO output tri-state buffer is enabled, a checksum is calculated, the packet is byte stuffed, the packet in memory is assembled and the DLL state is changed to FR_SEND. The formatted packet is sent using repeated calls to the dlSendAndReceiveByte function. If the current state is not IDLE then nothing happens.

With regard to the function plTXFinish it acts as a message to the DLL from the physical layer. When the byte received is a start byte then this corresponds to plStartByte otherwise this corresponds to plByteArrive.

The function plWriteCol acts as a message from the physical layer. It is called when there has been a write collision and handles the correction of that collision. The DLL state is changed to IDLE.

The function InitDataLinkLayer is called and reset as part of the processor initialisation. In order to initialise the data link layer, the timer is stopped, the data link layer buffer and global variables are cleared and the timeout values are intialised.

The slave session layer 205 is a communications protocol layer that functions to link configuration, link management and handle errors. The session layer 205 receives messages from the slave application, the slave DLL and the master session layer. The following functions act as messages to the slave session layer 205 from QSPI device specific firmware:

appSendPacket and appDisconnect.

The first mentioned function, appSendPacket, acts as a message from the application layer to the session layer. As this is a slave, this function is called when responding to a command or message from the master application layer (QSPI Driver in the controller 10). This function is a more general packet transmission function that can be used for transmitting any packet type from the QSPI device. It does not have any return value types and the arguments include a four bit network address, a four bit packet type, a data message length being the length of the data portion plus any

command bytes of the packet and finally the command bytes together with the data in the packet.

The function appDisconnect acts as a message from the application layer to the session layer and allows a logical disconnection from the master controller 10 so that the connection may be re-initialised. It has a network address as an argument and no return value types.

The following functions act as messages to the session layer 205 from non-QSPI device specific firmware:

sesAnyResp
sesReqInfoSet
sesConfirmCon
sesTIC
dlTimeout
dlPcktBad
dlCollision
dlPcktReceived

The functions sesAnyResp, sesReqInfoSet and sesConfirmCon act as messages from the master session layer to the slave session layer and are used in auto configuration. None of these functions have any arguments or return value types.

The function dlPcktBad acts as message from the datalink layer to the session layer. This message is sent if there is a received packet with a detected error. The data link layer is then changed to the idle state. The function has no arguments or a return value type.

The function dlCollision acts as a message from the data link layer to the session layer. The message is sent if there is a received packet, with a detected Write Collision. The data link layer is then changed to the idle state. This function has no arguments or a return value type.

The function dlPcktRecieved acts as a message from the data link layer to the session layer. This function is called by the data link layer when a valid packet has arrived. The packet is passed to the session layer for processing and the function implements the appropriate actions as described previously in relation to the QSPI communications protocol specification. This function sends the error checked packet to the application layer by passing a pointer to it. It has not return value types and it has two arguments one of which relates to the length of the packet received and the other being a pointer to the data that has been received.

The session layer has one timer that is used for timeouts and this is preferably an eight bit timer.

The slave application layer is a QSPI Device Application and has for its purpose to meet the functional requirements of a specific QSPI Device that is using the QSPI communications protocol to receive and process commands from the controller 10.

Therefore the function of the application layer is to receive commands from the controller 10, perform the desired action and respond.

As mentioned previously the sesPcktArrive is essentially a message to the slave application layer. It comprises sub-components appData which is a general data packet appCMD which is an example of a QSPI device specific command, being a sensor command packet, and appDisconnect which is a command from the master controller.

The function ProcessDevicePacket is called by the session layer to process the sesPcktArrive message. The three sub-messages above are processed as part of the sesPcketArrive function and hence in ProcessDevicePacket.

With regard to the function ProcessDevicePacket it has two arguments, one of which relates to the length of the packet received and the other being a pointer to the data that has been received. It has no return value types. All the commands and messages from the master application layer are handled by this function which decodes the packet and takes the appropriate action, and then sends the correct response/data. This function first looks at packetData[0]. The most significant nibble indicates the network address and the least significant nibble indicates the packet type. If the network address is equivalent to the current network address of the QSPI Device then the packet processing continues. If the packet type is a specific QSPI Device command, for example, sensor command appCMD, then the following byte indicates which command is being sent. If the packet type is a general data packet, such as appDATA, then a simple response packet is sent back. Any other packet type is not processed by the ProcessDevicePacket function.

Assuming that a QSPI device command has been sent then the function packetData[1] is read and compared to the enumerated type that defines the various commands for that device. The appropriate command is then executed and an appropriate response packet is returned using the sesPcktSend function in the SPI session layer. When responding a different packet type is used.

The InitDevice function has no arguments nor a return value type. This function is called at reset from the Processor Initialisation function. It performs any initialisation of the QSPI Device that is device dependent.

The function DoBackgroundTask also has no arguments or a return value type. It performs any device specific background task in the context of the main thread 204. Any communications processing will be executed in preference to this function in each loop. This function will be called continually and will perform better if it does not tie up the processor for too long at each call. While it is executing, new bytes that have been received cannot be processed.

The following description relates to the QSPI communication protocols between the controller 10 and each of the QSPI Devices. As mentioned previously the communications protocol consists of three layers being the physical layer, the data link layer and the session layer. Each of these layers provide a means for the master application in the controller 10 to communicate with the QSPI Device Application.

The physical layer is the SPI link between all of the sensors and the controller 10. It communicates between the controller or master 10 and one QSPI device or slave at a time. All the communications are controlled centrally by the controller 10.

The data link layer provides sending and receiving of packets data between the controller 10 and each of the slaves.. Error detection is carried out and communication timeouts are detected.

The session layer provides for the connection configuration and error handling. It allows the controller to send commands and receive QSPI Devices data. This layer provides the communications interface to the QSPI Device hardware application and the QSPI driver on the slaves and the controller respectively.

The navigator application is the QSPI driver which interrogates each of the sensors to obtain the QSPI device information and to control the QSPI Device operation.

Finally the QSPI Device application is a software module in the QSPI Device micro-controller 46 that receives commands from the controller 10 and carries them out.

Shown in Figure 22 are the various protocol layers on the controller side and the QSPI Device side and the data flow between the various layers. Shown in Figure 23 is a block diagram showing the various messages between each of the layers for the master controller and the various slave devices. In the QSPI Devices or slaves, the messages are implemented as functions. In the controller the messages between layers are implemented as functions with the result return.

As mentioned previously the session layer provides for the functions link management, link configuration and error handling in transmission. In addition to these it provides for the functions autoconfiguration, sending the application commands

to the correct QSPI Device, handling heartbeat messages and Identification messages as well as handling programming commands.

Programming in QSPI Devices is carried out by the controller 10. All the QSPI Devices must first be connected logically and programming mode is entered when a particular message is sent to the session layer. A command is then sent to the correct QSPI Device to start programming. The remaining devices are put into a "sleep" mode while this is occurring. The programming mode is exited by sending a particular message to the session layer when in programming mode.

Generic QSPI Device Commands are sent by the master session layer to the slave session layer. These commands have a packet type of "Configuration Packet". The slave session layer then handles the commands and returns the required information. The particular programming commands are handled by the session layer. All of the QSPI Device commands are triggered by a message from the application on the master or controller side to the session layer. The session layer then sends the specific command to the slave session layer which in turn carries out the desired action and returns the message if necessary.

With regard to the auto configuration function previously referred to it includes a procedure whereby a list of expected sensors in certain positions record the QSPI Device information about each QSPI Device. This includes, the lead that the QSPI device is plugged into, the serial number and the logical address, the logical connection status whether it is true or false.

Furthermore there is a quick configuration procedure which is an alternative to the automatic configuration procedure. The quick configuration procedure requires that the serial number of the QSPI Device being connected is known in advance. This procedure is for configuring a single QSPI device.

With reference to Figure 24 there is shown the state diagram that shows the general format of sending commands to a QSPI Device. When the session layer is idle, a message is received from the application layer to send a command to a specific QSPI Device, in this case the command appCMD. The state changes to SEND and then the appropriate packet is sent to the data link layer, using the message sesPcktSend, where it will be formatted and sent to the appropriate QSPI device.

When the packet has finished being sent, a dlPcktFinished message is received from the data link layer and the state changes to READ. The session layer then sends a message to the data link layer to read a packet from the QSPI bus, using the message sesPcktRead. When the data link layer has finished reading the packet and the packet

is correctly formatted without any transmission errors, a dlPcktReceived is received. The response is passed back to the application layer.

If there is an error of some kind, for example a time out, bad checksum or write collision then the session layer tries to send the command N times, currently, N = 5. If the command is still unsuccessful then a sesDeviceBad message is passed up to the application layer.

With reference to Figure 25 there is shown a state diagram concerning the QSPI Identification Command. It shows a situation where a command that is the responsibility of the session layer is used. A message is received from the application layer to read some identification information from a specific QSPI Device (Serial number, QSPI Device number, software version), the session layer then sends a command to the slave session layer via the lower layers of the protocol and the slave session layer processes the command and, if necessary, responds. A heartbeat pulse may also be requested to make sure the QSPI Device is still connected and functioning.

Shown in Figure 26 is a state diagram relating to the session layer in the slave QSPI Device. There are two main states the QSPI will be in and that is "not connected" and "connected". These refer to a logical connection between the master and the slave where information has been exchanged so that the master device knows where to send packets and the specific devices that are connected. At reset, a QSPI will be "not connected" and in an idle state. A portion of the diagram on the left side of Figure 26 describes the slave side of the automatic configuration procedure. Once the QSPI Device has been logically connected the state becomes C_IDLE. The session layer is now connected and waiting for a command.

If a packet arrives that is addressed to this QSPI Device then the data link level will send a message to the session layer with the packet that has arrived. If the packet contains a command which must be executed in the session layer then it is executed and, if necessary, a response is sent back via the sesPcktSend message to the data link layer.

If the arriving packet contains a command not processed by the session layer then it is passed to the application layer for processing and the state changes to ACTION. When the application layer has finished the command then it will send a message to the session layer to send a response back. Upon receipt of that message, the session layer sends the packet back to the master controller via the sesPcktSend message to the data link layer. When in the non-connected idle state NC_IDLE, if a sesAllRespondNow command is received, then the QSPI Device responds immediately with a predefined packet that is identical for all QSPI devices.

With regard to the data link layer this allows variable length packets of data to be sent and received to and from the controller and read from the slave devices. The packets will have eight-bit checksum error detection and in the event of an error a dlPcktBad message is sent to the next layer above. The checksum is calculated and appended to the message so that when the checksum of the message and checksum byte is calculated it should be 0. The checksum is calculated on the length and data bytes only.

Shown in Figure 27 is a state diagram that describes the data link layer in the slave QSPI Device. The overall function of the data link layer is to encode or decode a variable length sequence of bytes into a packet for transmission or reception. The DLL processes incoming and outgoing messages byte by byte and does byte stuffing and calculates the checksum.

The packet suitable for transmission over the physical layer has a start byte, which lets the data link layer know when a packet starts. A specific value is used for this function. As a consequence, the situation where another part of the packet happens to be the same value as the start byte must be considered. When this occurs, a single start byte inside the message is replaced with two start bytes, the reverse occurring at the other end.

The events entailed in receiving the packet are:

The DLL is in IDLE;

The DLL knows a packet is starting to arrive as a start byte has just arrived, that is message plStartByte. The state then changes to FR_START.

If the next byte is a start byte as well then the length is this value or the packet is actually starting at that moment. In this case the state changes to STUFF1. The next byte indicates the length. The state then changes to REC_N having received the length byte. A counter is started to count the bytes being received so that the DLL knows when the packet is finished. The length byte is written to a buffer.

If a non-start byte arrives then the state changes to INC_N. The counter is incremented and the byte is written to the buffer. If a start byte is received then the state changes to STUFF2. If the next byte received is a start byte also then that single byte in the packet was indeed a start byte and the state changes to INC_N, incrementing the counter.

If the next byte was not a start byte then it is assumed that the packet is starting again and that byte is actually the length and the state and changes to REC_N. The process then moves to INC_N. If a start byte is received then the state changes to STUFF3 and back to INC_N for the next start byte and the counter is incremented by

one. For non-start bytes, the state remains in INC_N unless the last byte in the packet has been read. When the last byte in the packet has been read then the checksum must be checked to see if there have been any transmission errors and the state changes to CRC_CHK. When the checksum has been calculated either a "Packet Bad" message (dlPcktBad) is sent or a "Packet Received" message, that is dlPcktReceived, is sent to the session layer. The state changes to IDLE when the packet has been received.

In the case of transmitting a packet, the DLL is in the IDLE state and receives a sesPcktSend message from the session layer requesting that a packet is to be sent. The DLL then calculates the checksum and makes the packet, byte stuffing where necessary. Then the state changes to FR_SEND. The counter is set to the packet length. As the slave QSPI Device cannot initiate a serial transfer of a byte itself, it waits for the master controller to read the bytes in the packet. When a byte is sent from the controller 10 that is not a start byte, that is, a packet is not being sent from the controller 10, the SEND state is entered, the counter is decremented and the next byte is copied to the SPI bus for transmission.

When all of the bytes have been transmitted the state returns to IDLE. If a start byte is received while the slave is transmitting a packet then this means that the controller 10 is transmitting a packet also and must take precedence. Thus the state then changes to FR_START and the DLL moves into a receiving packet mode.

With reference to Figure 28 there is shown a state diagram that describes the operation of the DLL in the controller 10. In principle this is the same as the slave layer DLL but varies due to the difference in the way communication is initiated from the controller 10 for both sending and receiving.

Receiving a packet is initiated by a sesPcktRead message from the session layer which causes the state to change to POLLING. The controller 10 then actively reads the SPI bus until a start byte is received. Then the remainder of the packet reception is the same as for the slave DLL except that the bytes read are initiated by the controller 10.

Sending a packet is initiated by a sesPcktSend message from the session layer which causes a transmission sequence similar to the slave layer DLL. The only difference being that a "Transmission Completed" message (plTxFinish) triggers the state transition to send the next byte.

It will be appreciated by persons skilled in the art that numerous variations and/or modifications may be made to the invention as shown in the specific embodiments without departing from the spirit or scope of the invention as broadly

described. The present embodiments are, therefore, to be considered in all respects as illustrative and not restrictive.

Dated this 16th day of April 2004

Neopraxis Pty Ltd
Patent Attorneys for the Applicant:

F B RICE & CO

5.4.6.3 Wiener Filtering

The problem of estimating the sagittal angle from the 2 angle estimation sources explained in Section 5.4.6.2 may be formulated as a problem of estimating a noise-like signal in noise with different spectral characteristics. The basic structure of the wiener angle is shown in Figure 5-4.

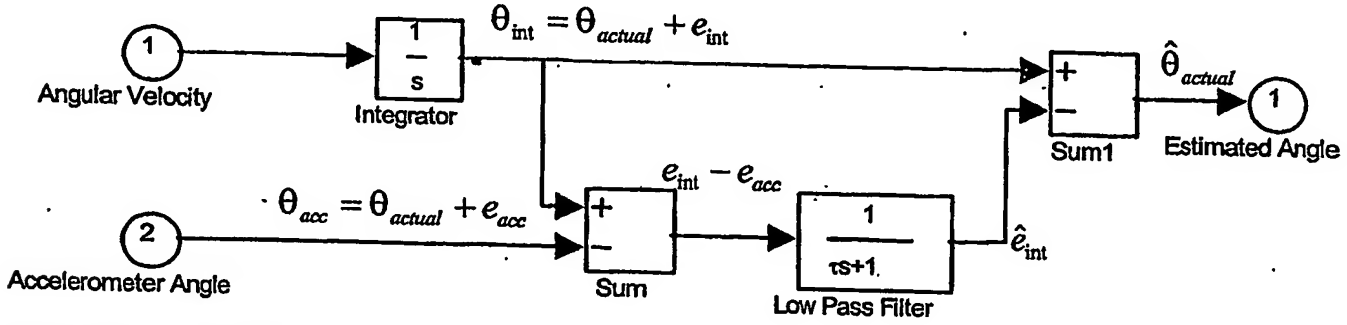

Figure 5-4 - Basic Wiener Estimator Structure

In Figure 5-4, the difference in the integrated angle measurement (θ_{int}) and the acceleration derived angle measurement (θ_{acc}) is calculated. This signal is therefore the difference in the errors in both these measurements. An assumption is made about the characteristics of the two errors with respect to their spectral content. The error in the integrated angle (e_{int}) is assumed low frequency noise. The error in the accelerometer-derived angle (e_{acc}) is assumed to be high frequency noise. Characterising e_{int} as low frequency noise accounts for the drift due to a small zero offset with the angular velocity measurement. Characterising e_{acc} as high frequency noise accounts for the error in the accelerometer angle being due to movement in the sensor with is an inherently changing signal. Figure 5-5 shows a representation of the spectral content of the e_{int} and $-e_{acc}$ signals.

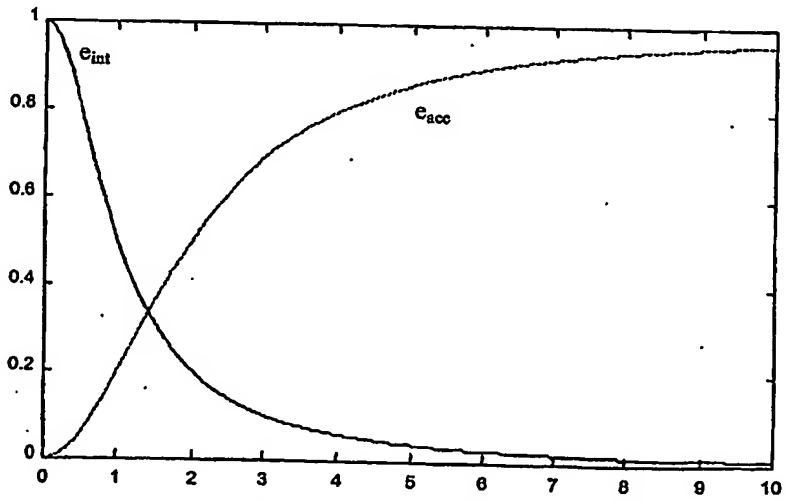

Figure 5-5 - Spectral Representation of the Angle estimation errors

A first order low pass filter (LPF) in series with a rate limiter is used to extract an estimate of e_{int} ($\hat{e}_{int}$) from the angle difference. A τ value of 0.6 is used in the LPF. This value of τ is chosen based on the assumption that error in the integrated angle is any component of the estimate difference signal (e_{int}-e_{acc}) below about 0.5Hz.

The rate limiter is set to one of two separate values depending on an estimation of how still the sensor is. A low

slew rate reduces the effect that the accelerometer derived angle has on the final angle. The slew rate is set to 100°/s when the sensor is approximately still and to 5°/s when the sensor is moving. When the magnitude squared of the acceleration vector is within 15% of g^2 and the magnitude of the angular velocity is less than 90°/s then the sensor is considered "still" for the purposes of adjusting the slew rate.

The final estimate of θ is hence given by:

$$\hat{\theta}_{actual} = \theta_{actual} + e_{int} - \hat{e}_{int} \qquad (5\text{-}7)$$

A further problem must be overcome due to the inevitable drift of the θ_{int} value that is internally calculated within the algorithm. Even a very small error in the calibration value for the zero point will result in large errors in θ_{int} internally over time. This could lead to the internal value of θ_{int} exceeding integer bounds of the variable used to store the value. This in turn could lead to stability problems in the estimator.

The effect of a potential overflow in the internal value θ_{int}, can be eliminated by the addition of 2 elements. The first element is a hysteresis at the output of the integrator. The second element is a modified angle difference function used to calculate the difference between angles. The hysteresis ensures that θ_{int} changes continuously throughout the range -250°<θ<250°. When either of these extremities is reached, the angle has 360° added or subtracted as appropriate to keep the angle within range. The modified angle difference function returns the difference between any two angles. The difference will always be in the range +180° to -180°. Hence any offset in an angle by a multiple of 360° will be ignored in the difference function.

There is a discontinuity in the accelerometer-derived angle θ_{acc} at ±180°. When the angle is near 180° or -180° noise on the accelerometer signal takes the calculated value of θ_{acc} over this discontinuity. The θ_{acc} signal then looks like very large magnitude noise as the angle swaps from 180° to -180°. This causes the sagittal estimation to be unreliable around this point. The difficulty using this angle estimate at this point of discontinuity can be resolved by noting that it is not the discontinuous angle measurement itself that is used for the rest of the sagittal estimation algorithm, but the difference between this estimate and the output of the integrator. Hence, using the modified difference function mentioned earlier, we can eliminate any effect that this 360° offset may have caused.

Hysteresis is put the estimated output angle. The hysteresis ensures that the output angle changes continuously throughout the range -250°<θ<250°. When either of these extremities is reached, the angle has 360° added or subtracted as appropriate to keep the angle within range.

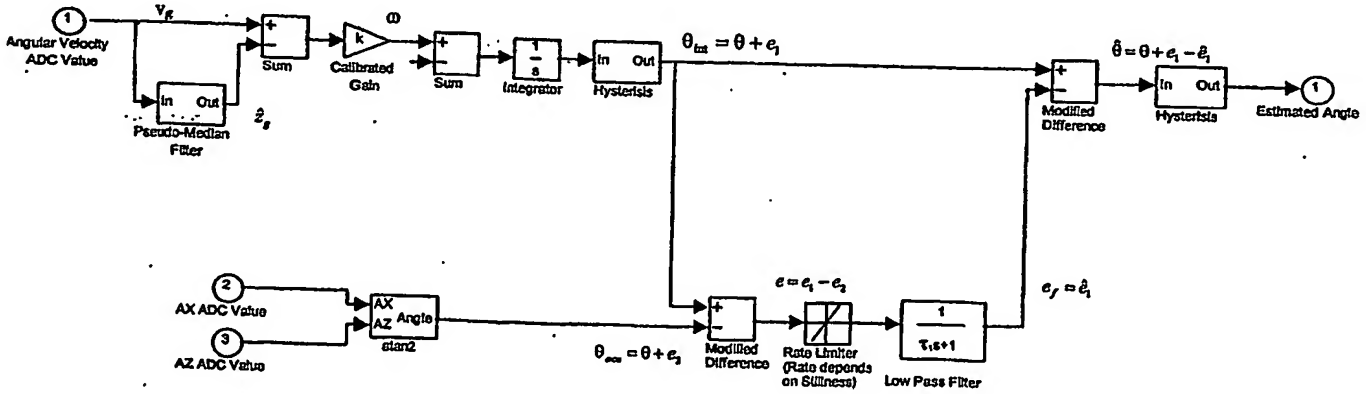

Figure 5-6 - Complete Angle Estimator

APPENDIX 2

PSEUDO-MEDIAN FILTER

1.1 Variables

Table 1—1

Variable	Description
M	Current Median Estimate
ε	Step size.
highCount	Count of samples higher than M+ε.
lowCount	Count of samples lower than M-ε.
equalCount	Count of samples between M-ε and M+ε.
sampleCount	Samples between median updates.
n	Current sample.
X	Input sample.

1.2 Algorithm

At each iteration of the sensor angle estimation, which runs as a background task, the following algorithm is executed to estimate the current ADC value that corresponds to a 0°/s reading from the gyroscope. A median filter over a large number of samples would be a good measure of this, however this is processor intensive. A first order LPF using integer arithmetic and a sufficiently high time constant has quantisation errors that have detrimental effects on the estimation. The following is a pseudo-median filter that attempts to estimate the current 0°/s ADC value. X is the gyroscope voltage reading, which is an integer between 0 and 1023.

1. At each iteration increment either highCount, lowCount, equalCount depending on the value of X with respect to M and ε.
2. Increment n
3. If n≥sampleCount (sampleCount is currently 128, which corresponds to approximately 1 second per update of M).
    a. Increase M by ε if the median is above M+ε, i.e. if highCount>(lowCount+equalCount). Also increment ε by 1.
    b. Decrease M by ε if the median is below M-ε, i.e. if lowCount>(highCount+equalCount). Also increment ε by 1.
    c. If either of the conditions in (a) and (b) are not true then the median remains unchanged and ε is decremented by 2.

2 GYROSCOPE CALIBRATION PROGRAM

The aim is to find a value of z_g and k so that

$$\omega = k(v_g - z_g) \tag{2-1}$$

Where:

ω is the angular velocity.

k is the gain.

v_g is the Gyroscope ADC Value.

z_g is the Gyroscope ADC Value corresponding to zero °/s.

This is achieved by rotating the sensor in the sagittal plane from a resting position to a different orientation (resting) over a period of 2 seconds. The sensor must be stationary at the start and end of this period. The ADC values from the sensor are sampled and stored at a frequency of 50Hz by the navigator. Once the sampling has taken place, the following steps are followed to calculate the gain and zero offset of the gyroscope.

Measure the start and finish angles using the ADC values and the calibration values already stored in the sensor pack.

We know that

$$\Delta\theta = \theta_f - \theta_i = k\int_0^t (v_g - z_g)dt \tag{2-2}$$

$$\Delta\theta = \left[\int_0^t v_g dt \quad \int_0^t dt\right]\begin{bmatrix} k \\ -k z_g \end{bmatrix} \tag{2-3}$$

Do n measurements. For the ith measurement:

$$y_i = \Delta\theta$$

$$x_i = \int_0^t v_g dt \tag{2-4}$$

Several measurements are taken for x_i, y_i, and T_i. A least squares best fit is used to calculate k and z_g.

$$\begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 & T_1 \\ \vdots & \vdots \\ x_i & T_i \\ \vdots & \vdots \\ x_n & T_n \end{bmatrix}\begin{bmatrix} k \\ -k z_g \end{bmatrix} \tag{2-5}$$

(2-5) can be rewritten:

$$\underline{y} = A\underline{b} \tag{2-6}$$

This equation is solved in a least squares sense using:

$$A^T \underline{y} = A^T A \underline{b}$$
$$\underline{b} = \left(A^T A\right) A^T \underline{y}$$

so that: (2-7)

$$k = \frac{\left(\sum_{i=1}^{i=n} T_i^2\right)\left(\sum_{i=1}^{i=n} x_i y_i\right) - \left(\sum_{i=1}^{i=n} T_i x_i\right)\left(\sum_{i=1}^{i=n} T_i y_i\right)}{\left(\sum_{i=1}^{i=n} x_i^2\right)\left(\sum_{i=1}^{i=n} T_i^2\right) - \left(\sum_{i=1}^{i=n} T_i x_i\right)^2} \tag{2-8}$$

$$z_g = \frac{\left(\sum_{i=1}^{i=n} T_i x_i\right)\left(\sum_{i=1}^{i=n} x_i y_i\right) - \left(\sum_{i=1}^{i=n} T_i y_i\right)\left(\sum_{i=1}^{i=n} x_i^2\right)}{\left(\sum_{i=1}^{i=n} T_i^2\right)\left(\sum_{i=1}^{i=n} x_i y_i\right) - \left(\sum_{i=1}^{i=n} T_i x_i\right)\left(\sum_{i=1}^{i=n} T_i y_i\right)} \tag{2-9}$$

The values of k and z_g are converted to calibration coefficients suitable for use by the sensor pack. The angular velocity in the Sensor Pack is calculated using the formula:

$$\omega\left(8°/s\right) = \frac{8192\left(v_g - z_g\right)}{K_d} \tag{2-10}$$

So K_d can be calculated using $K_d = \dfrac{8192}{8°/s \bullet \left(-k\right)}$

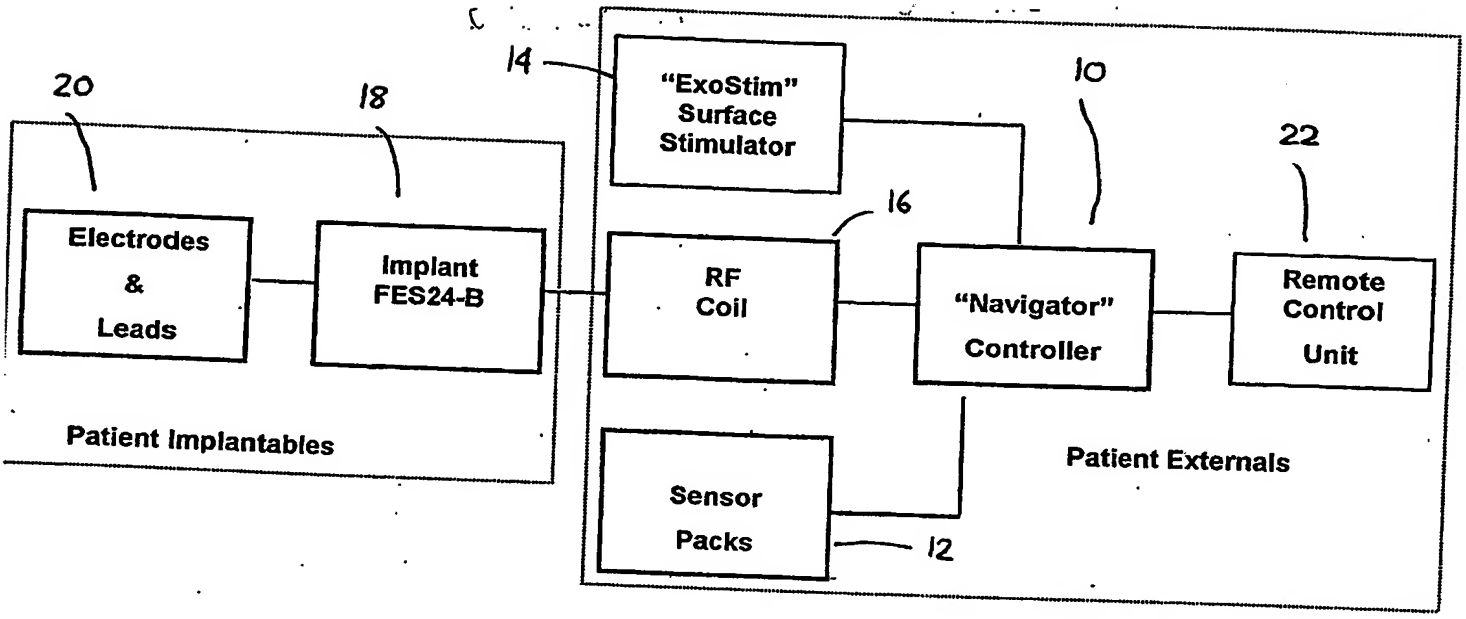
20
18
14
"ExoStim"
Surface
Stimulator
10
22
16
Electrodes
&
Leads
Implant
FES24-B
RF
Coil
"Navigator"
Controller
Remote
Control
Unit
Patient Implantables
Patient Externals
Sensor
Packs
12

FIG. 1

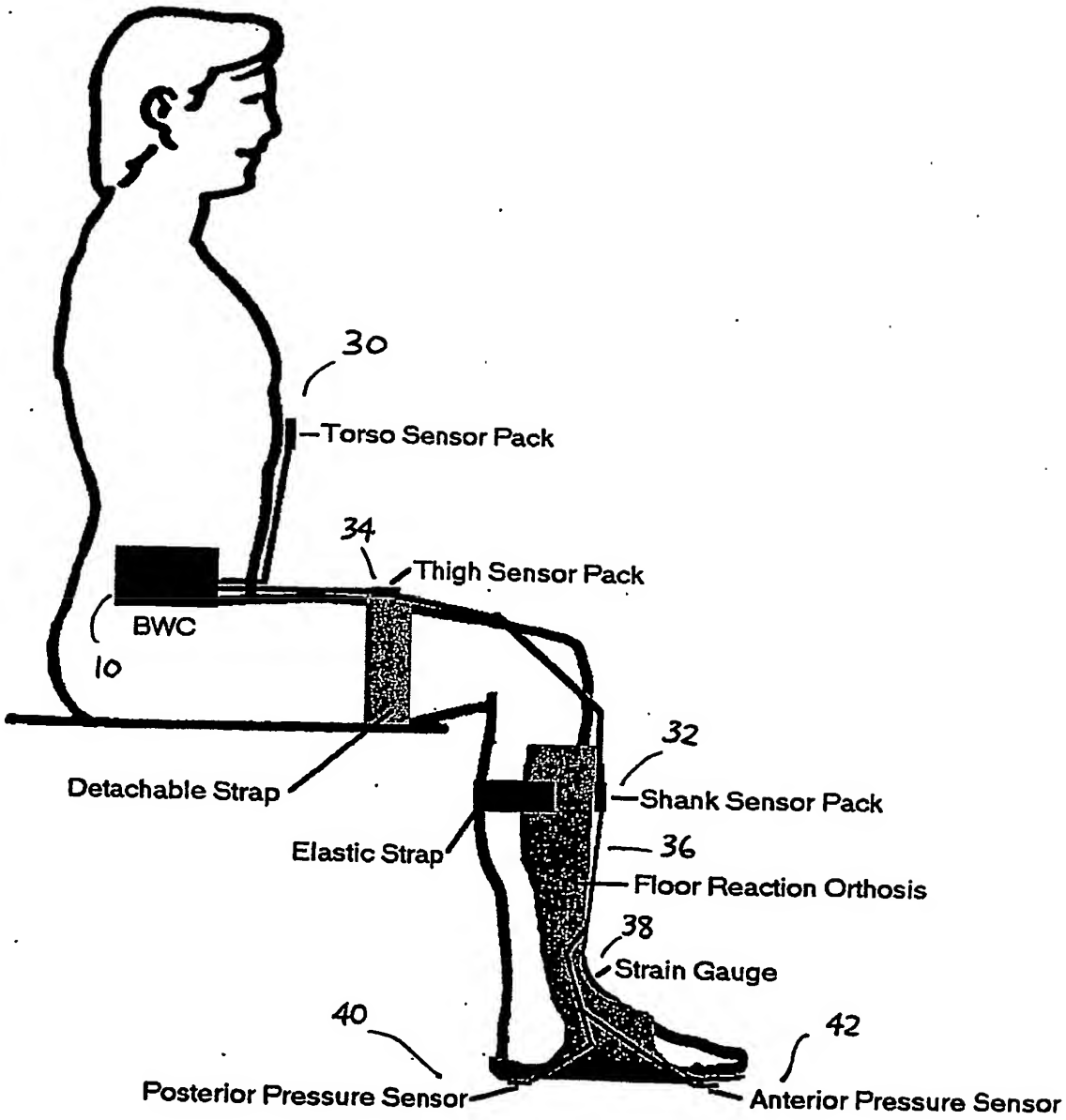
30
Torso Sensor Pack
34
Thigh Sensor Pack
BWC
10
32
Shank Sensor Pack
Detachable Strap
Elastic Strap
36
Floor Reaction Orthosis
38
Strain Gauge
40
42
Posterior Pressure Sensor
Anterior Pressure Sensor

FIG. 2

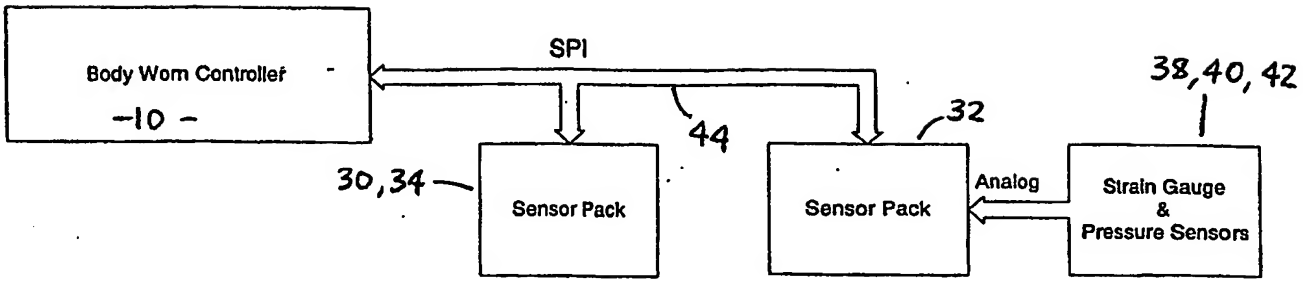
Body Worn Controller
-10-
SPI
44
38,40,42
32
30,34
Sensor Pack
Sensor Pack
Analog
Strain Gauge
&
Pressure Sensors

FIG. 3A

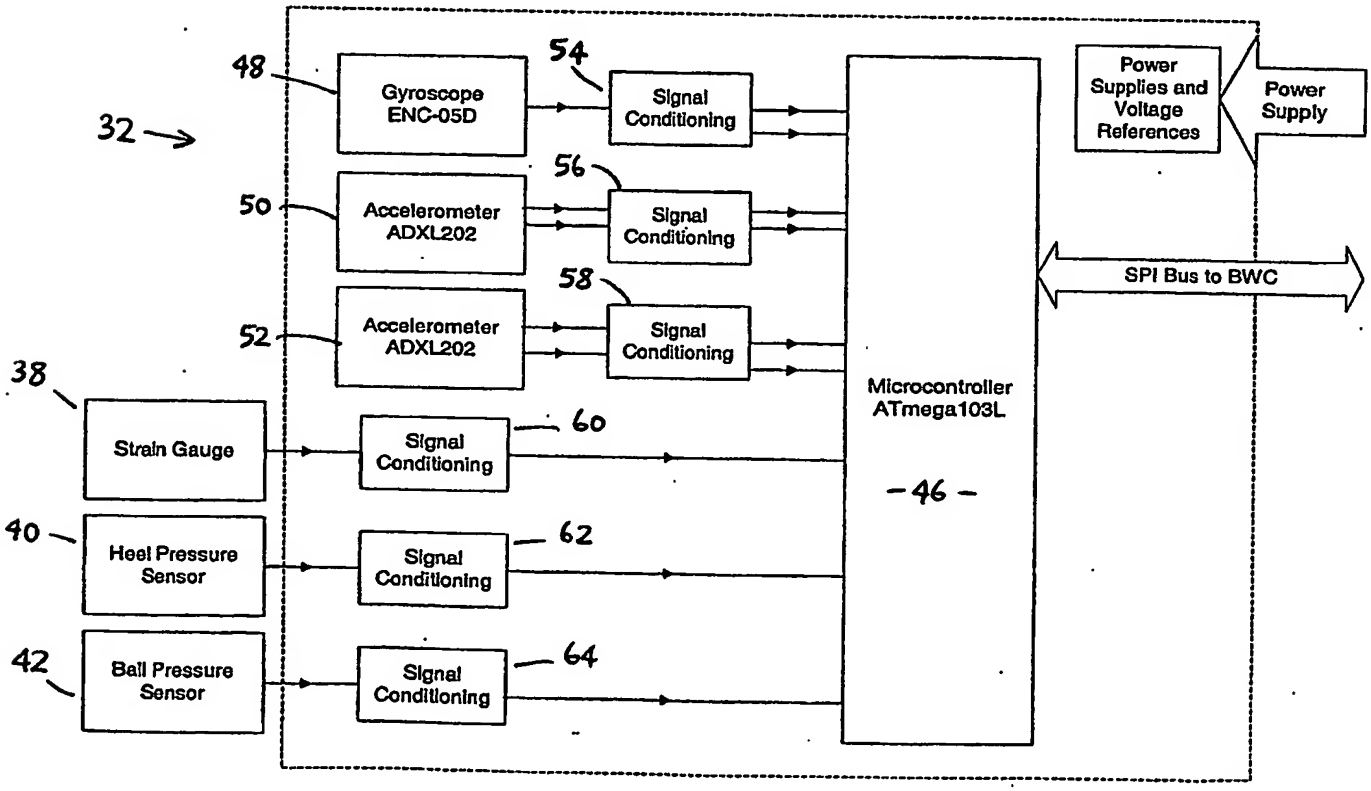
32
48
Gyroscope
ENC-05D
54
Signal
Conditioning
50
Accelerometer
ADXL202
56
Signal
Conditioning
52
Accelerometer
ADXL202
58
Signal
Conditioning
38
Strain Gauge
Signal
Conditioning
60
40
Heel Pressure
Sensor
Signal
Conditioning
62
42
Ball Pressure
Sensor
Signal
Conditioning
64
Microcontroller
ATmega103L
-46-
Power
Supplies and
Voltage
References
Power
Supply
SPI Bus to BWC

FIG. 3B

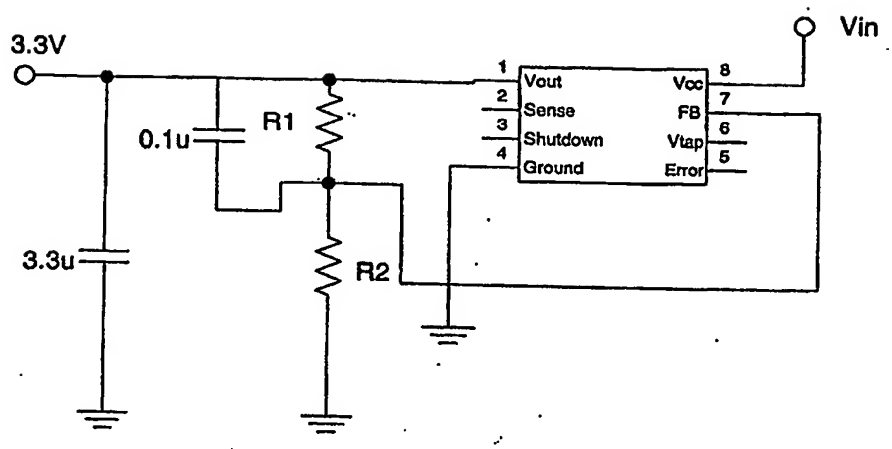
3.3V
Vin
0.1u
R1
3.3u
R2
1 Vout
2 Sense
3 Shutdown
4 Ground
8 Vcc
7 FB
6 Vtap
5 Error

FIG. 4

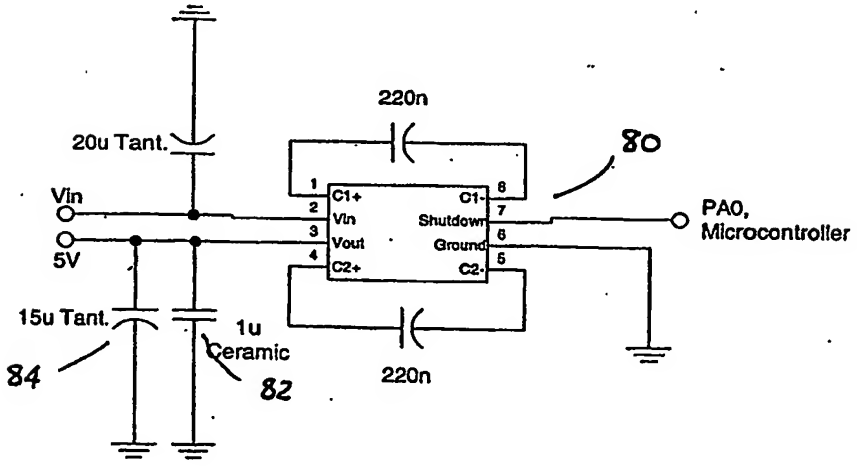
220n
20u Tant.
80
Vin
5V
1 C1+
2 Vin
3 Vout
4 C2+
8 C1-
7 Shutdown
6 Ground
5 C2-
PA0,
Microcontroller
15u Tant.
1u
Ceramic
84
82
220n

FIG. 5

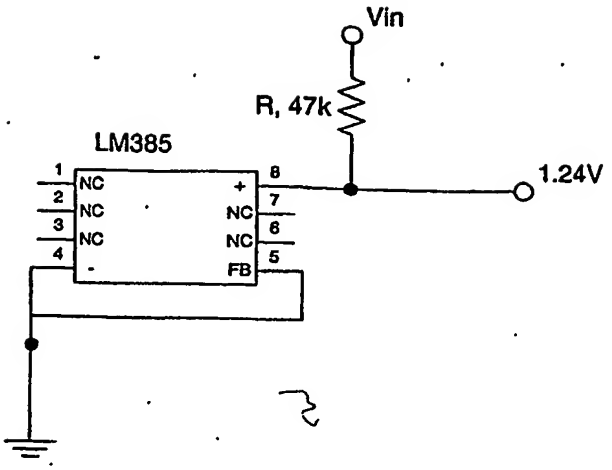

FIG. 6

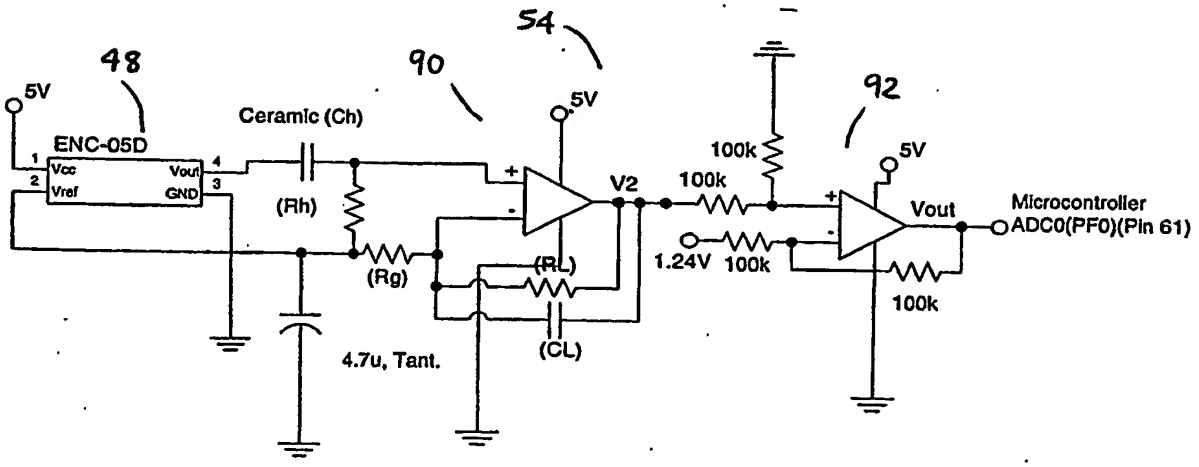

FIG. 7

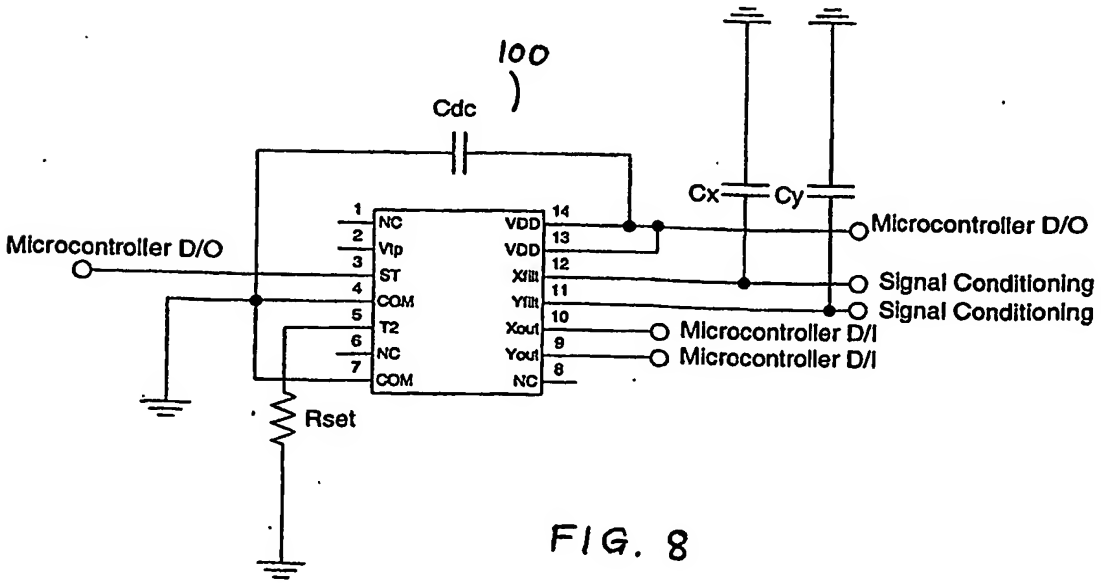

FIG. 8

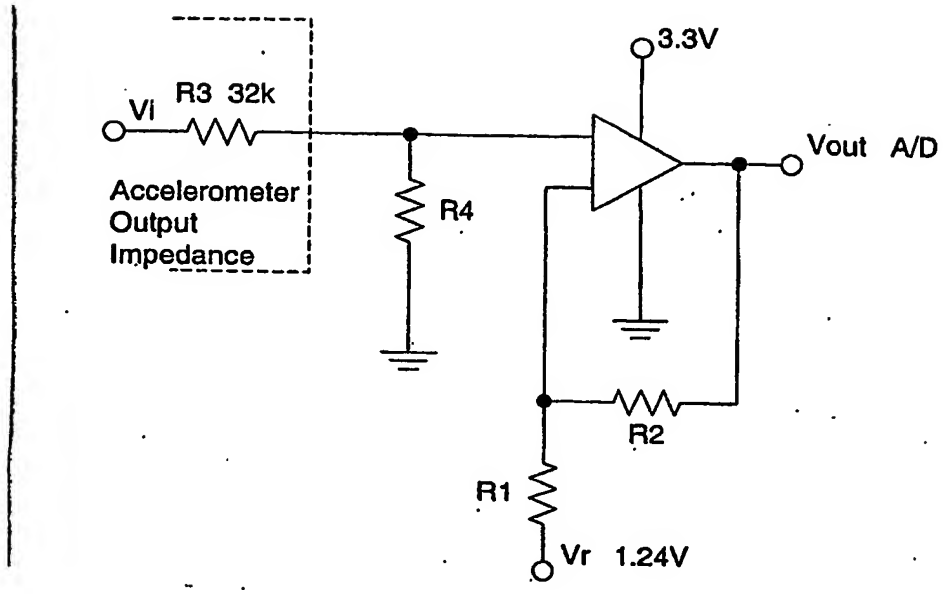
3.3V
Vi
R3 32k
Vout A/D
Accelerometer
Output
Impedance
R4
R2
R1
Vr 1.24V

FIG. 9

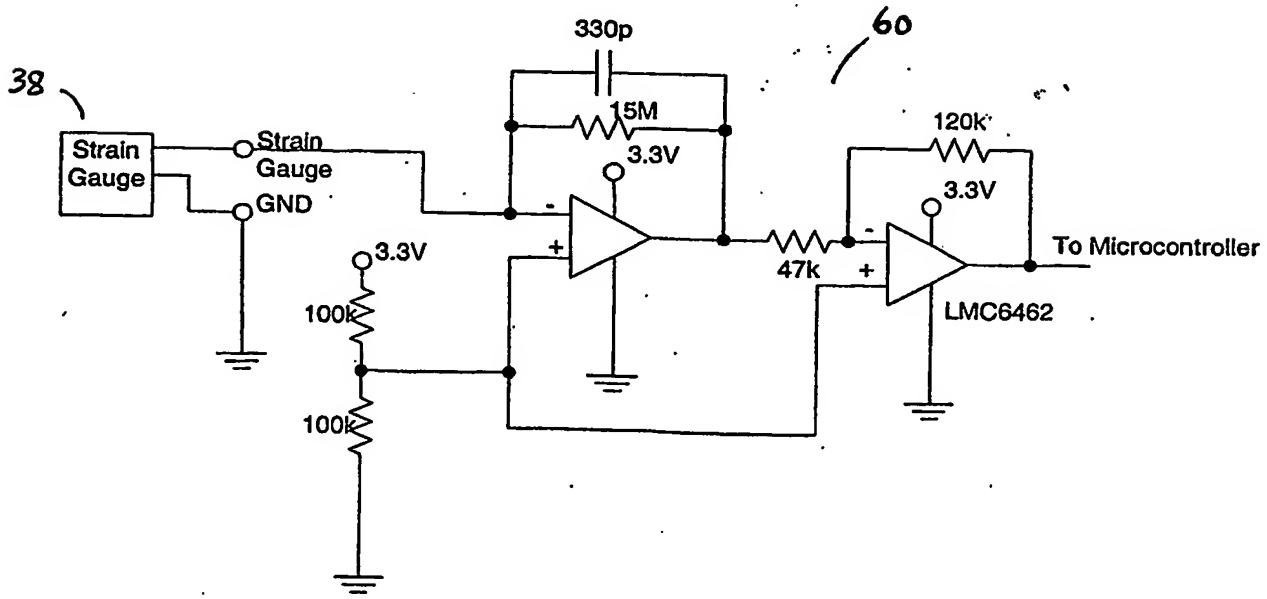
38
60
330p
15M
120k
Strain
Gauge
Strain
Gauge
GND
3.3V
3.3V
3.3V
100k
100k
47k
To Microcontroller
LMC6462

FIG. 10

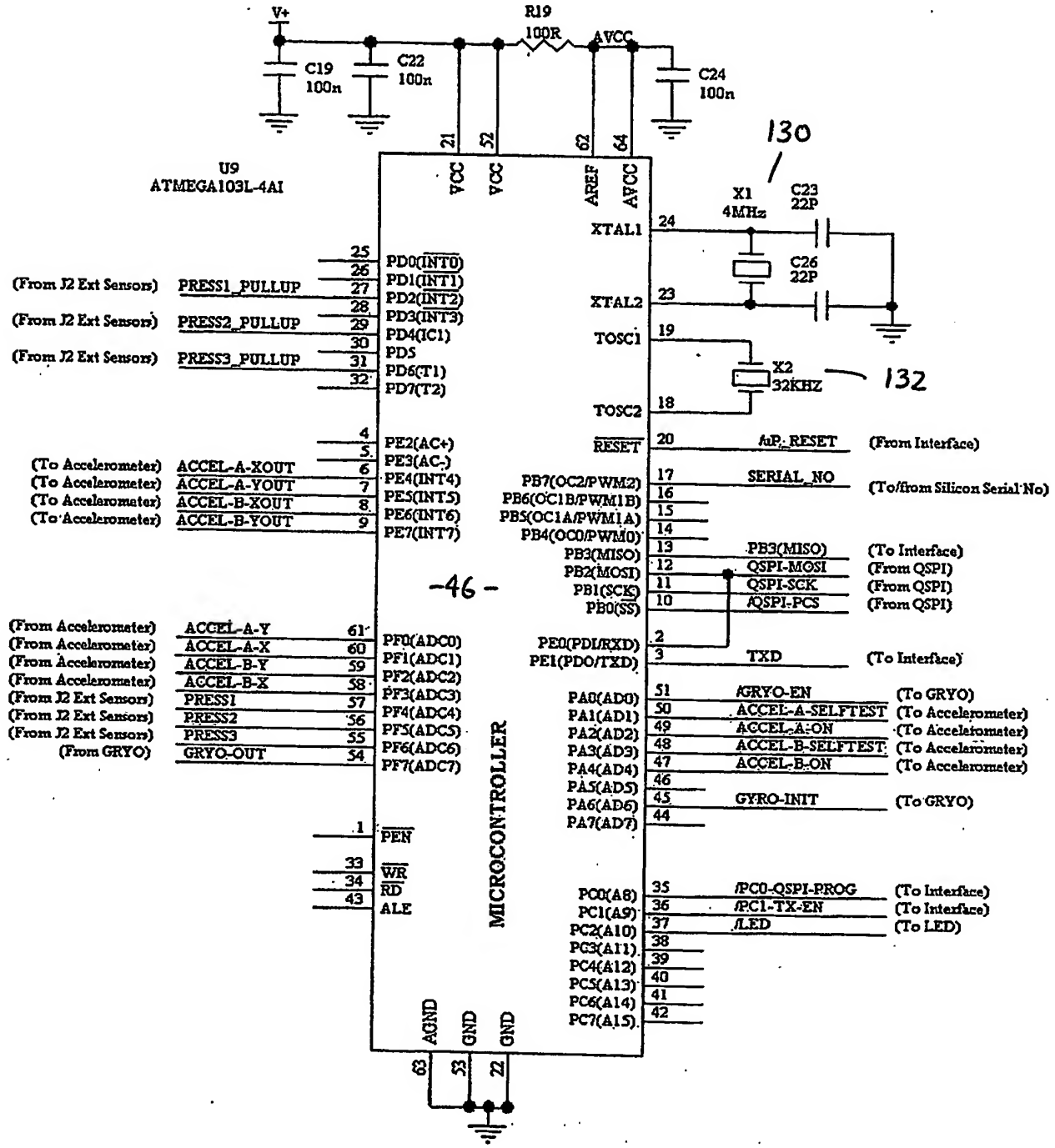
V+
R19
100R
AVCC
C19
100n
C22
100n
C24
100n
130
U9
ATMEGA103L-4AI
VCC
VCC
AREF
AVCC
X1
4MHz
C23
22P
C26
22P
X2
32KHZ
132
XTAL1
XTAL2
TOSC1
TOSC2
RESET
PD0(INT0)
PD1(INT1)
PD2(INT2)
PD3(INT3)
PD4(IC1)
PD5
PD6(T1)
PD7(T2)
PE2(AC+)
PE3(AC-)
PE4(INT4)
PE5(INT5)
PE6(INT6)
PE7(INT7)
PB7(OC2/PWM2)
PB6(OC1B/PWM1B)
PB5(OC1A/PWM1A)
PB4(OC0/PWM0)
PB3(MISO)
PB2(MOSI)
PB1(SCK)
PB0(SS)
PF0(ADC0)
PF1(ADC1)
PF2(ADC2)
PF3(ADC3)
PF4(ADC4)
PF5(ADC5)
PF6(ADC6)
PF7(ADC7)
PE0(PDI/RXD)
PE1(PDO/TXD)
PA0(AD0)
PA1(AD1)
PA2(AD2)
PA3(AD3)
PA4(AD4)
PA5(AD5)
PA6(AD6)
PA7(AD7)
PEN
WR
RD
ALE
PC0(A8)
PC1(A9)
PC2(A10)
PC3(A11)
PC4(A12)
PC5(A13)
PC6(A14)
PC7(A15)
AGND
GND
GND
MICROCONTROLLER
-46-
(From J2 Ext Sensors) PRESS1_PULLUP
(From J2 Ext Sensors) PRESS2_PULLUP
(From J2 Ext Sensors) PRESS3_PULLUP
(To Accelerometer) ACCEL-A-XOUT
(To Accelerometer) ACCEL-A-YOUT
(To Accelerometer) ACCEL-B-XOUT
(To Accelerometer) ACCEL-B-YOUT
(From Accelerometer) ACCEL-A-Y
(From Accelerometer) ACCEL-A-X
(From Accelerometer) ACCEL-B-Y
(From Accelerometer) ACCEL-B-X
(From J2 Ext Sensors) PRESS1
(From J2 Ext Sensors) PRESS2
(From J2 Ext Sensors) PRESS3
(From GRYO) GRYO-OUT
/uP_RESET (From Interface)
SERIAL_NO (To/from Silicon Serial No)
PB3(MISO) (To Interface)
QSPI-MOSI (From QSPI)
QSPI-SCK (From QSPI)
/QSPI-PCS (From QSPI)
TXD (To Interface)
/GRYO-EN (To GRYO)
ACCEL-A-SELFTEST (To Accelerometer)
ACCEL-A-ON (To Accelerometer)
ACCEL-B-SELFTEST (To Accelerometer)
ACCEL-B-ON (To Accelerometer)
GYRO-INIT (To GRYO)
/PC0-QSPI-PROG (To Interface)
/PC1-TX-EN (To Interface)
/LED (To LED)

FIG. 11

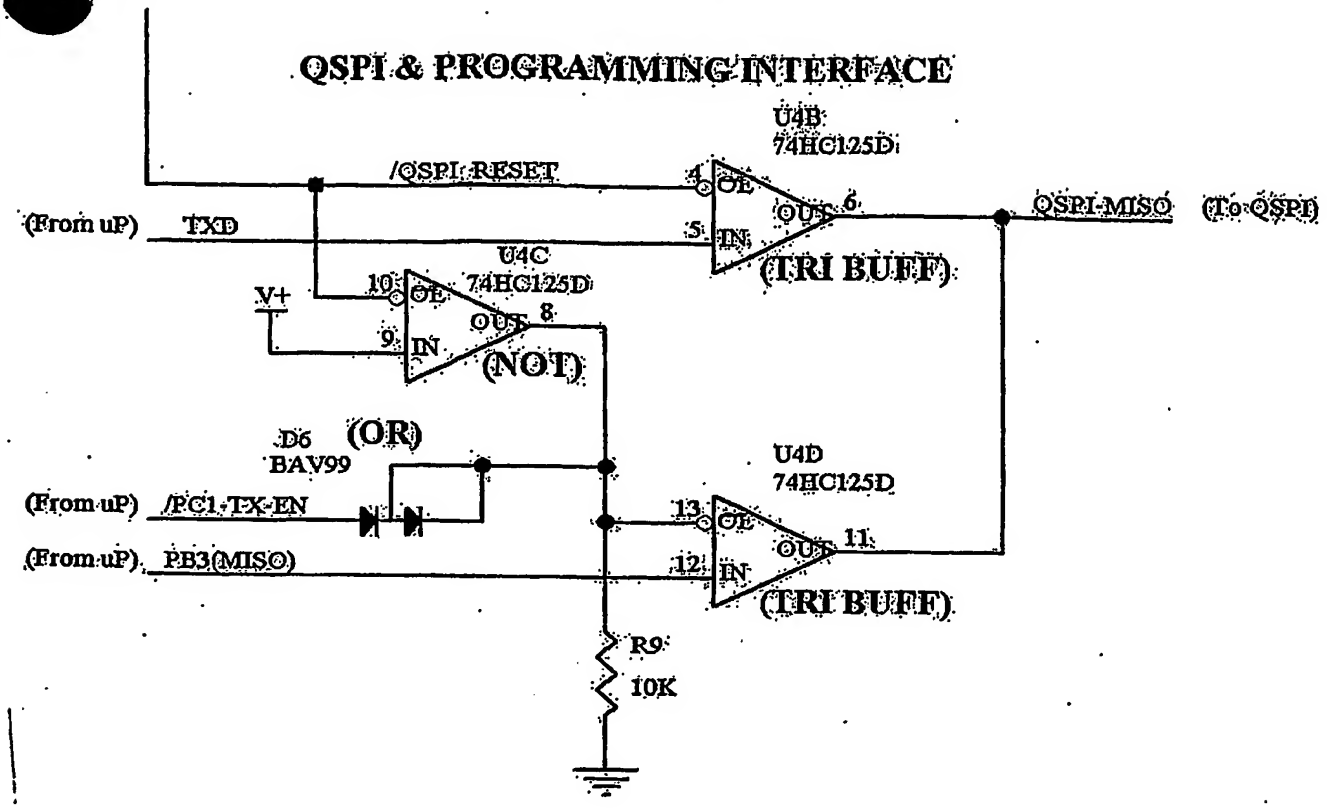

FIGURE 11A

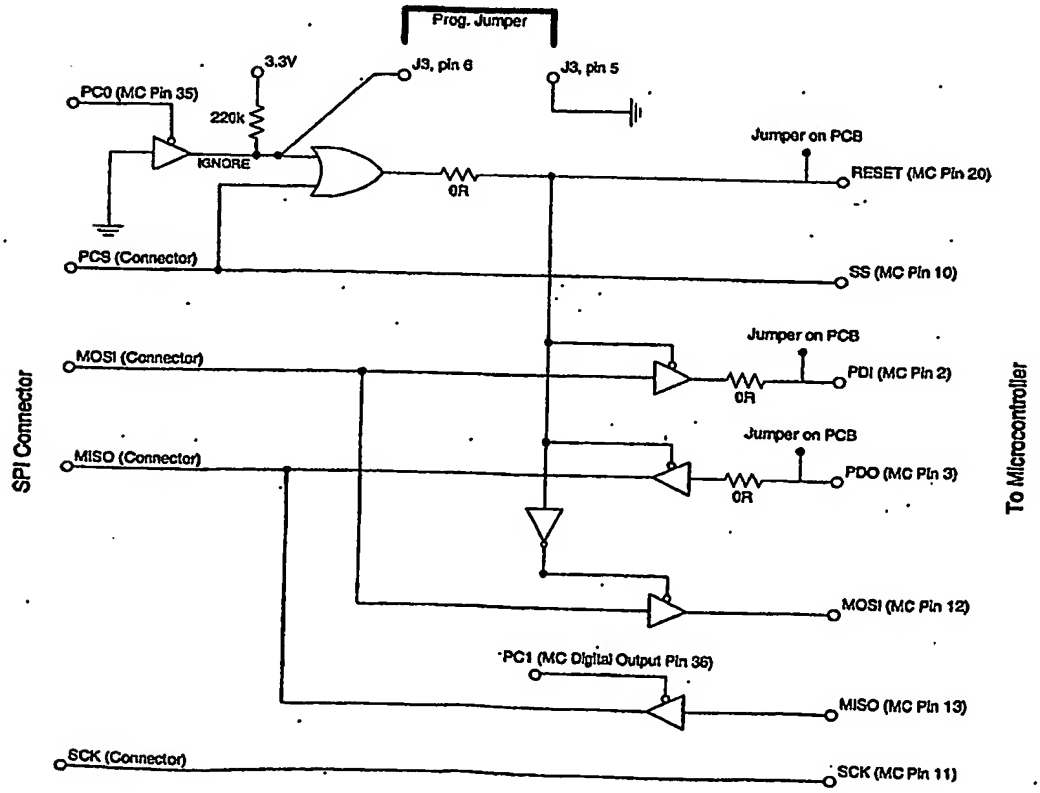

FIGURE 11B

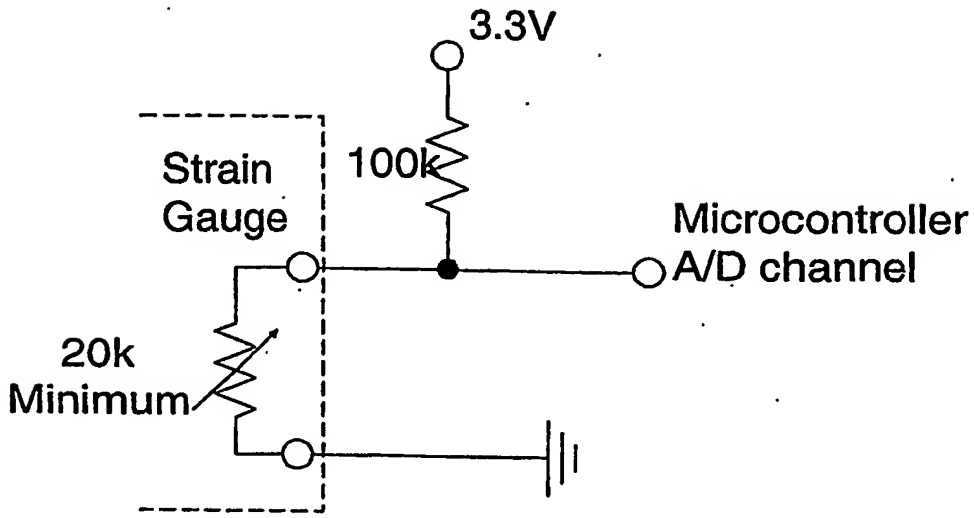
3.3V
Strain Gauge
100k
Microcontroller A/D channel
20k Minimum

FIG. 12

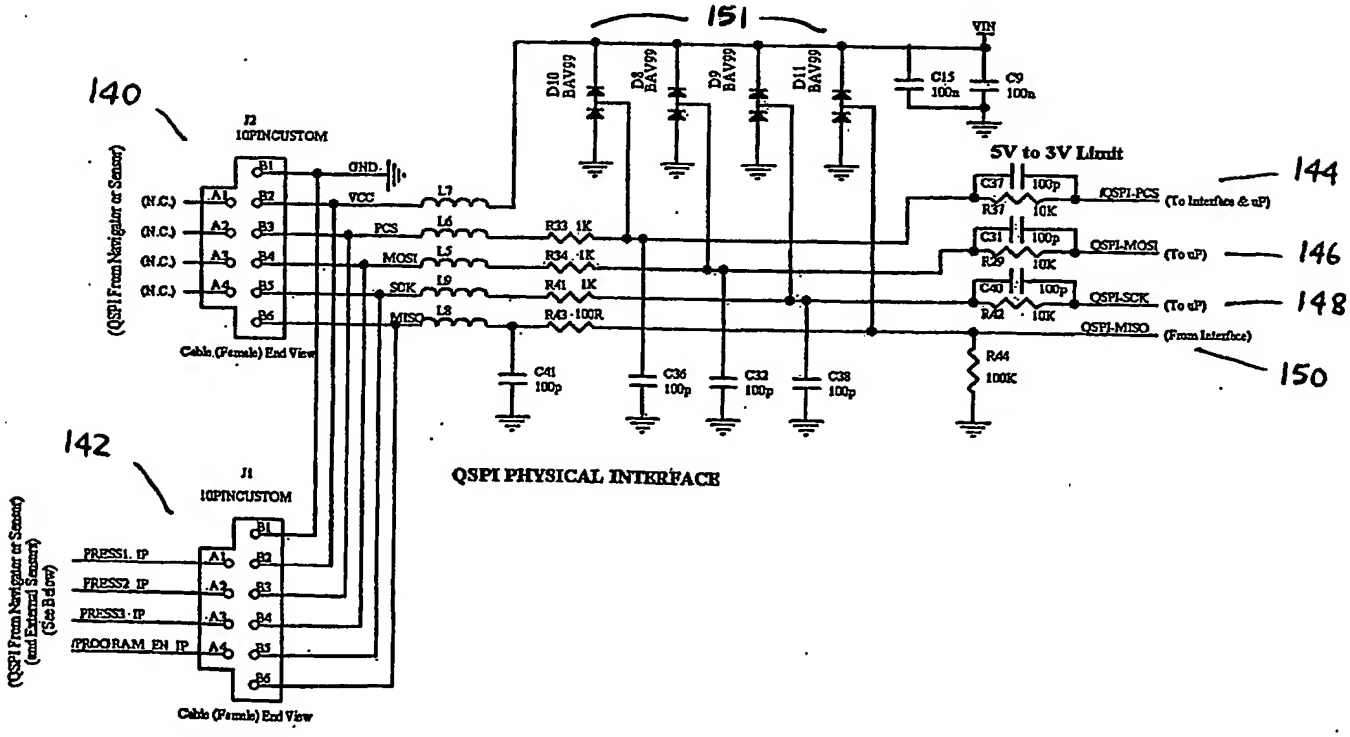
151
140
142
144
146
148
150
J2
10PINCUSTOM
J1
10PINCUSTOM
(QSPI From Navigator or Sensor)
Cable (Female) End View
GND
VCC
PCS
MOSI
SCK
MISO
(N.C.)
PRESS1_IP
PRESS2_IP
PRESS3_IP
PROGRAM_EN_IP
D10 BAV99
D8 BAV99
D9 BAV99
D11 BAV99
C15 100n
C9 100n
VIN
5V to 3V Limit
C37 100p
R37 10K
/QSPI-PCS
(To Interface & uP)
C31 100p
R29 10K
QSPI-MOSI
(To uP)
C40 100p
R42 10K
QSPI-SCK
(To uP)
QSPI-MISO
(From Interface)
R33 1K
R34 1K
R41 1K
R43 100R
R44 100K
C41 100p
C36 100p
C32 100p
C38 100p
QSPI PHYSICAL INTERFACE

FIG. 13

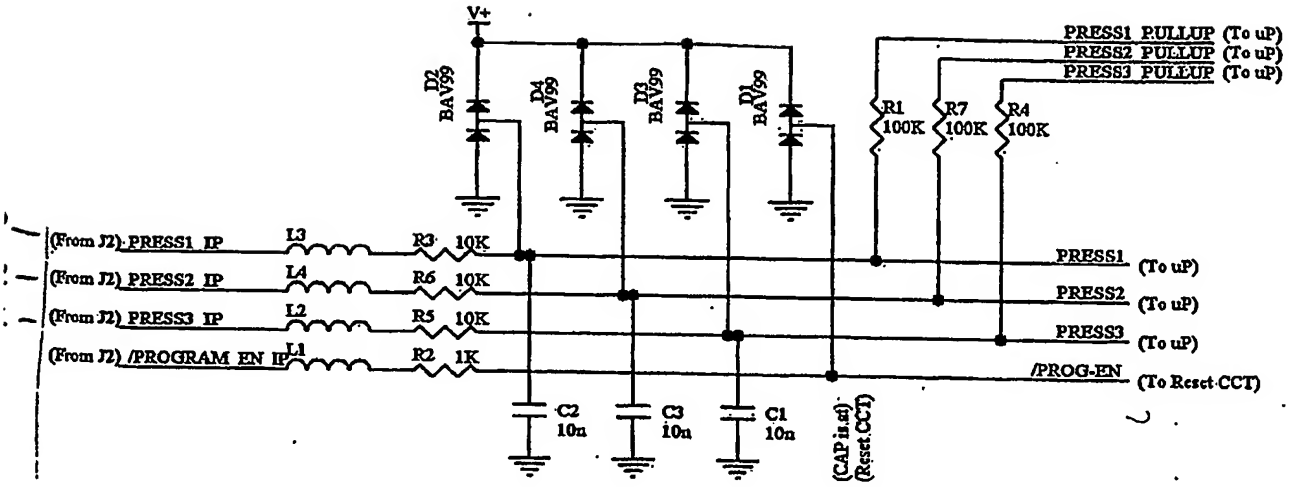
V+
D2 BAV99
D4 BAV99
D3 BAV99
D1 BAV99
R1 100K
R7 100K
R4 100K
PRESS1 PULLUP (To uP)
PRESS2 PULLUP (To uP)
PRESS3 PULLUP (To uP)
(From J2) PRESS1 IP
(From J2) PRESS2 IP
(From J2) PRESS3 IP
(From J2) /PROGRAM EN IP
L3
L4
L2
L1
R3 10K
R6 10K
R5 10K
R2 1K
PRESS1 (To uP)
PRESS2 (To uP)
PRESS3 (To uP)
/PROG-EN (To Reset CCT)
C2 10n
C3 10n
C1 10n
(CAP is at) (Reset CCT)

FIG. 14

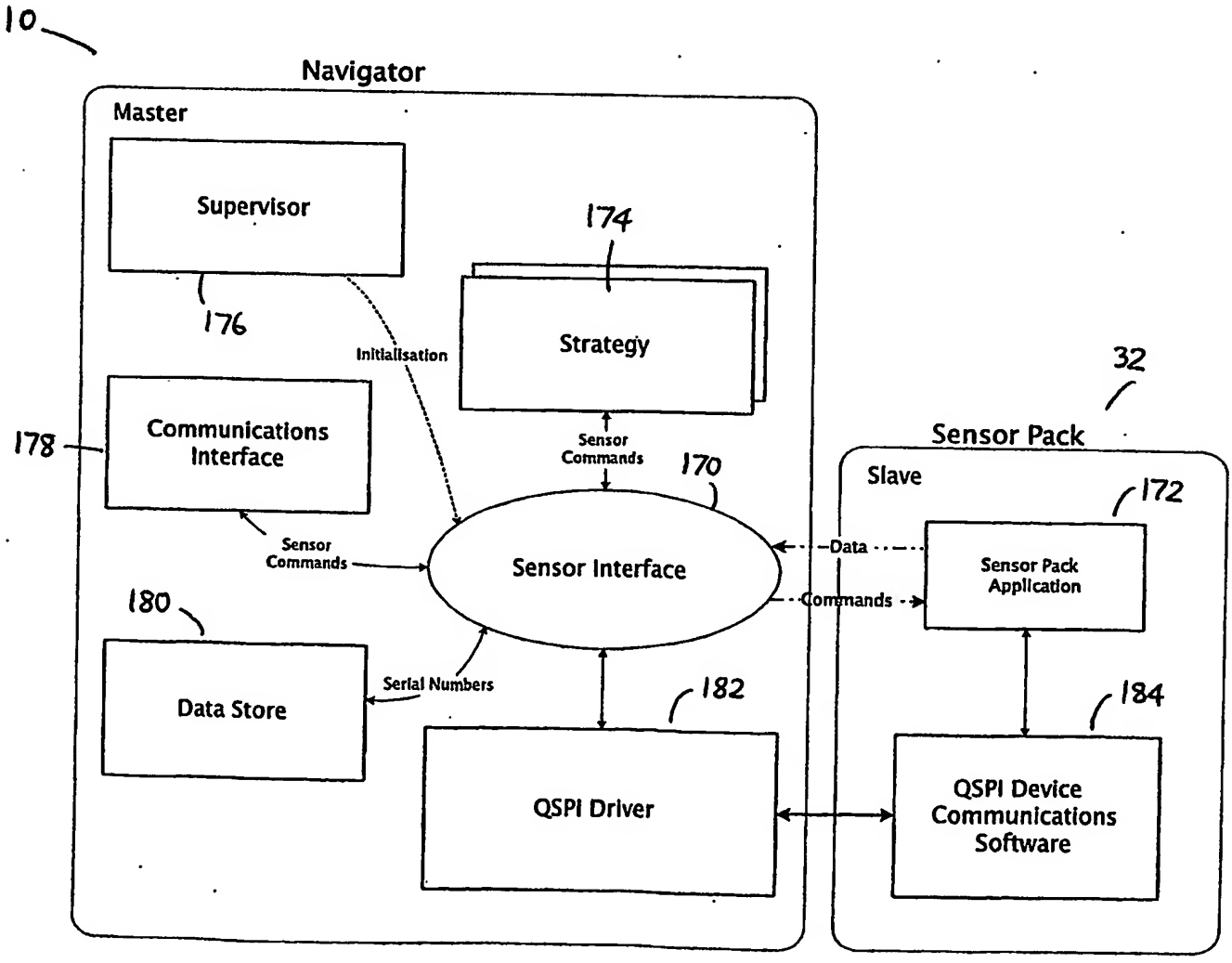
10
Navigator
Master
Supervisor
176
174
Strategy
Initialisation
178
Communications Interface
Sensor Commands
170
Sensor Commands
Sensor Interface
180
Data Store
Serial Numbers
182
QSPI Driver
32
Sensor Pack
Slave
172
Data
Commands
Sensor Pack Application
184
QSPI Device Communications Software

FIG. 15

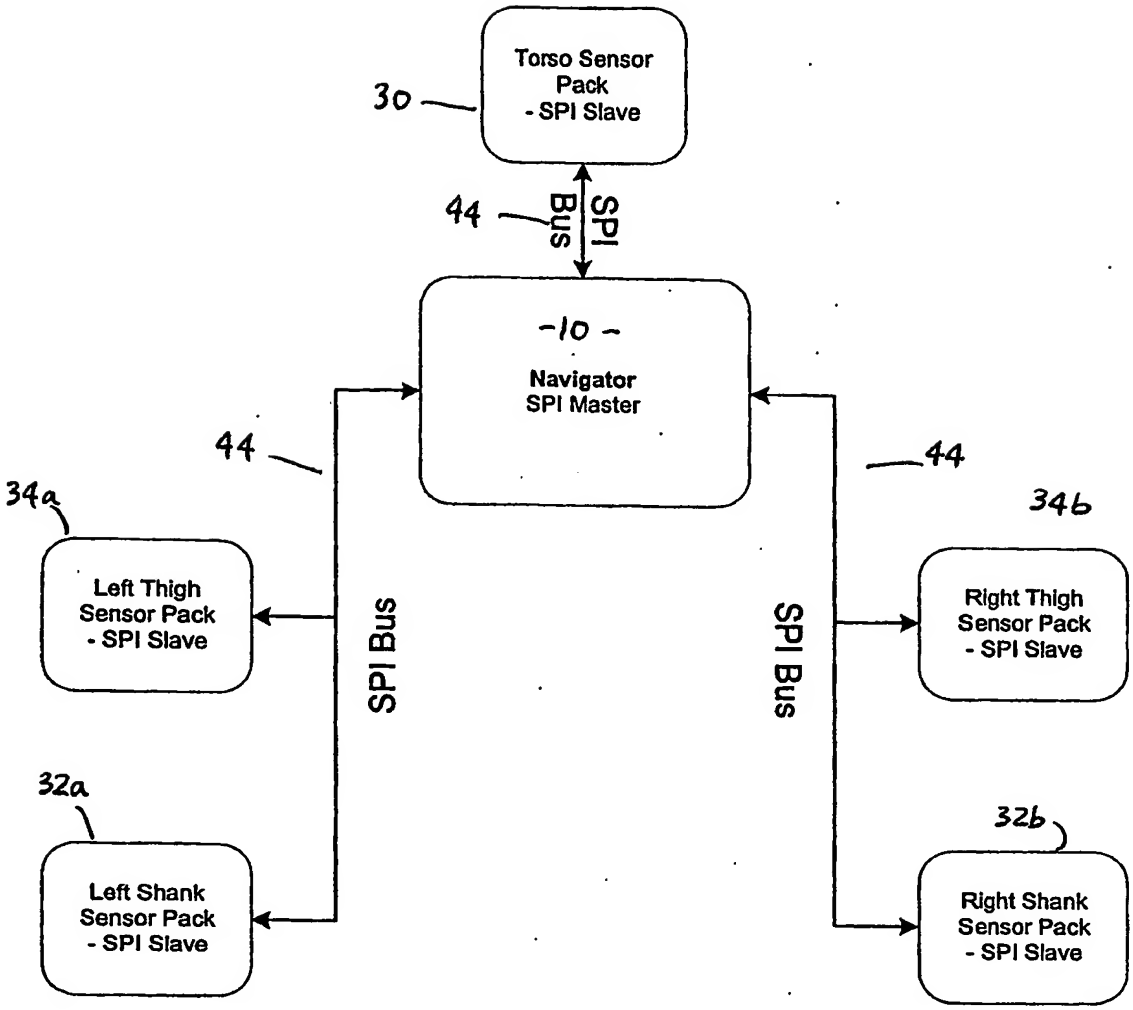
Torso Sensor Pack - SPI Slave
30
44
SPI Bus
-10-
Navigator SPI Master
44
44
34a
34b
Left Thigh Sensor Pack - SPI Slave
SPI Bus
SPI Bus
Right Thigh Sensor Pack - SPI Slave
32a
32b
Left Shank Sensor Pack - SPI Slave
Right Shank Sensor Pack - SPI Slave

FIG. 16

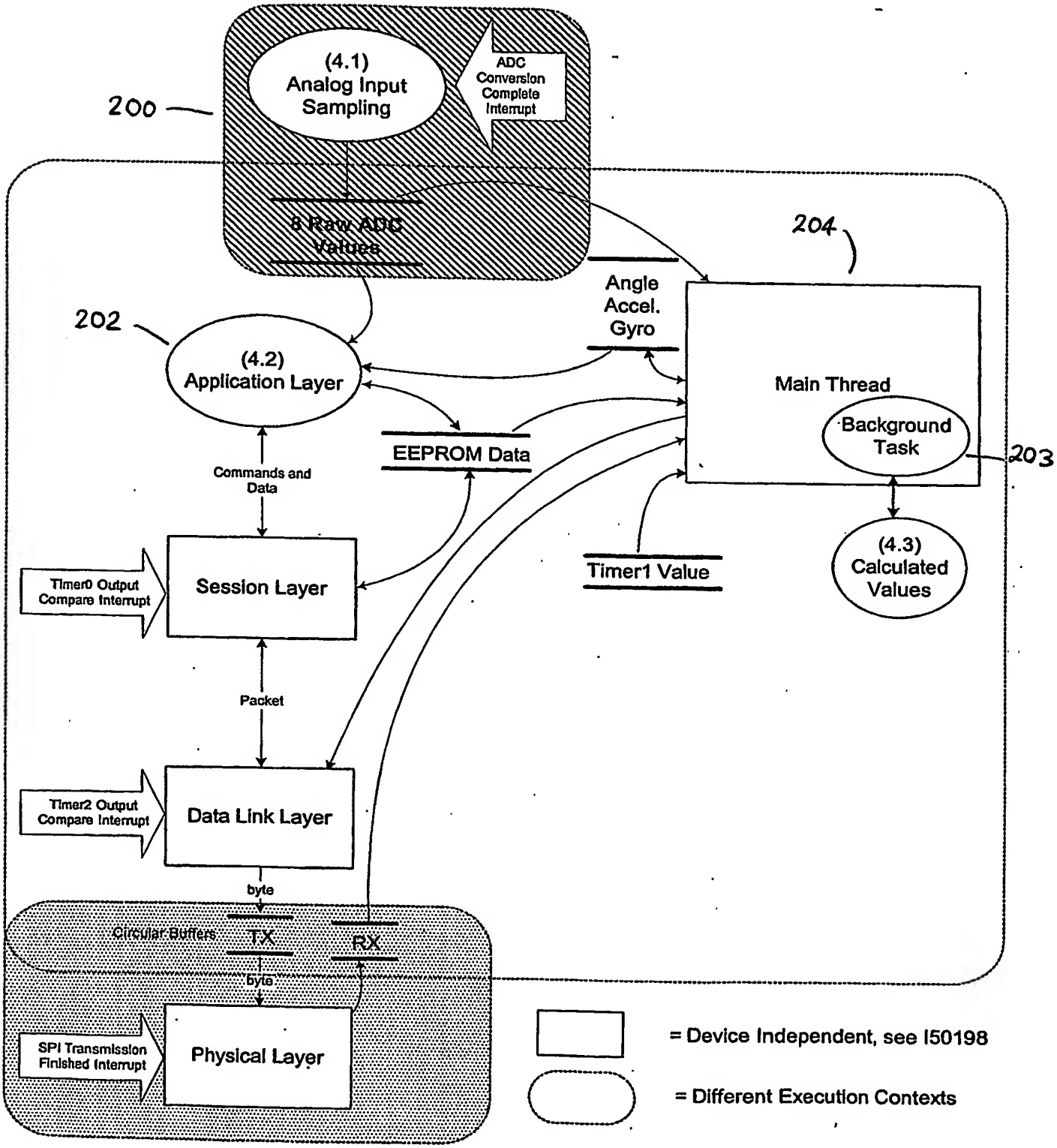
200
(4.1)
Analog Input
Sampling
ADC
Conversion
Complete
Interrupt
8 Raw ADC
Values
204
Angle
Accel.
Gyro
202
(4.2)
Application Layer
Main Thread
Background
Task
203
EEPROM Data
Commands and
Data
Timer1 Value
(4.3)
Calculated
Values
Timer0 Output
Compare Interrupt
Session Layer
Packet
Timer2 Output
Compare Interrupt
Data Link Layer
byte
Circular Buffers
TX
RX
byte
SPI Transmission
Finished Interrupt
Physical Layer
= Device Independent, see I50198
= Different Execution Contexts

FIG. 17

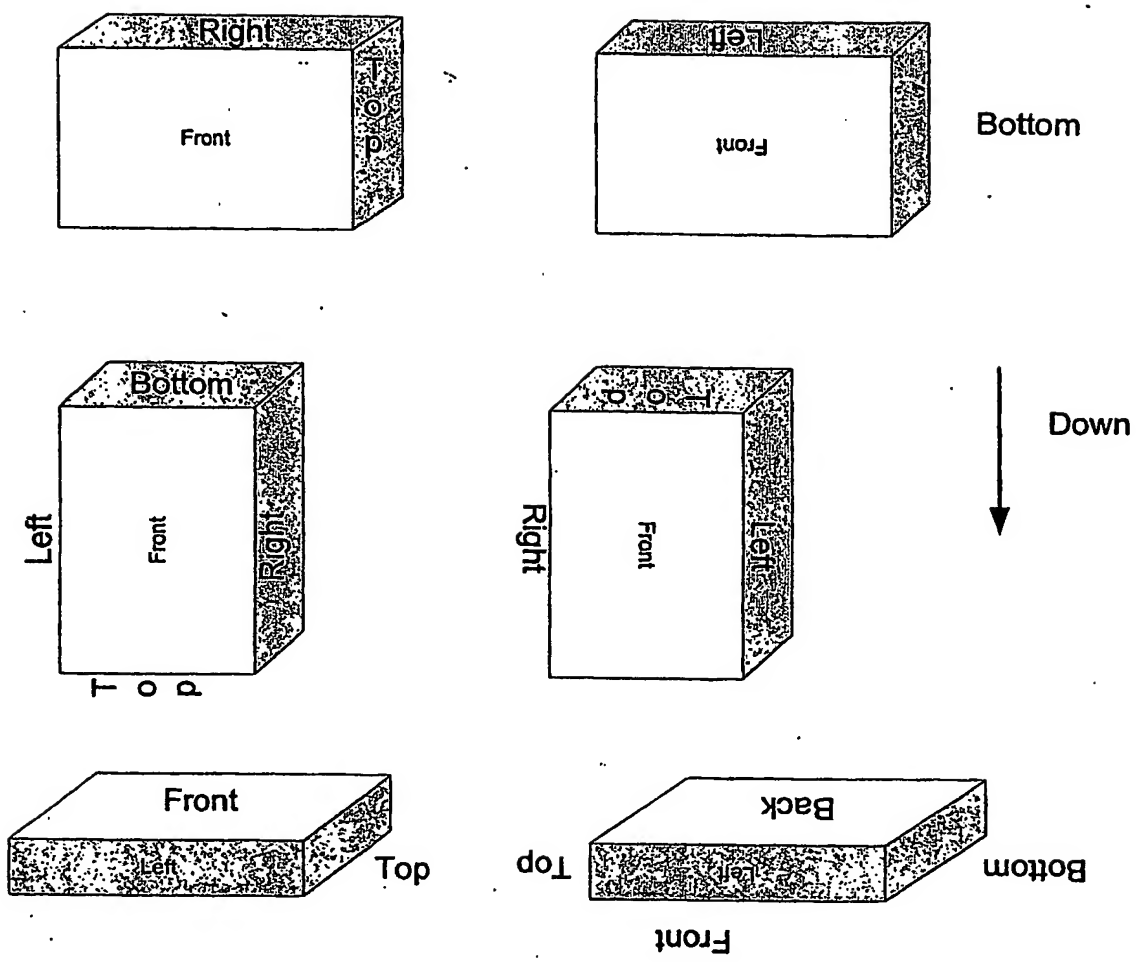
Right
Top
Front
Left
Front
Bottom
Bottom
Left
Front
Right
Top
Top
Front
Left
Right
Down
Front
Left
Top
Top
Back
Left
Bottom
Front

FIG. 18

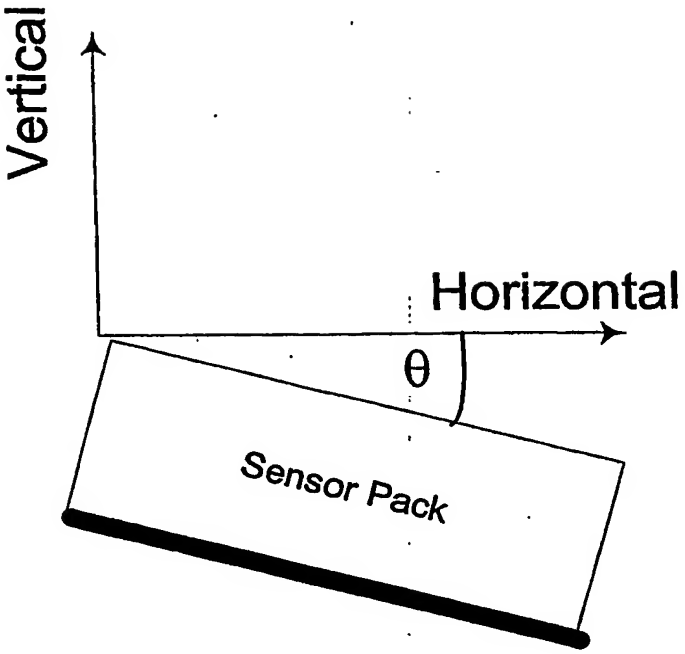
Vertical
Horizontal
θ
Sensor Pack

FIG. 19

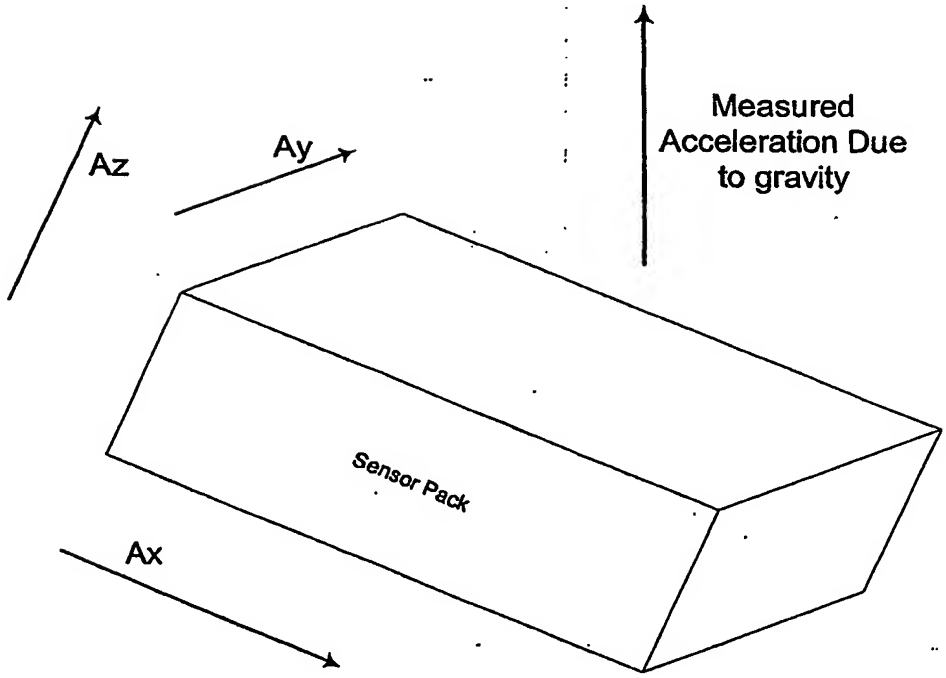
Az
Ay
Measured
Acceleration Due
to gravity
Sensor Pack
Ax

FIG. 20

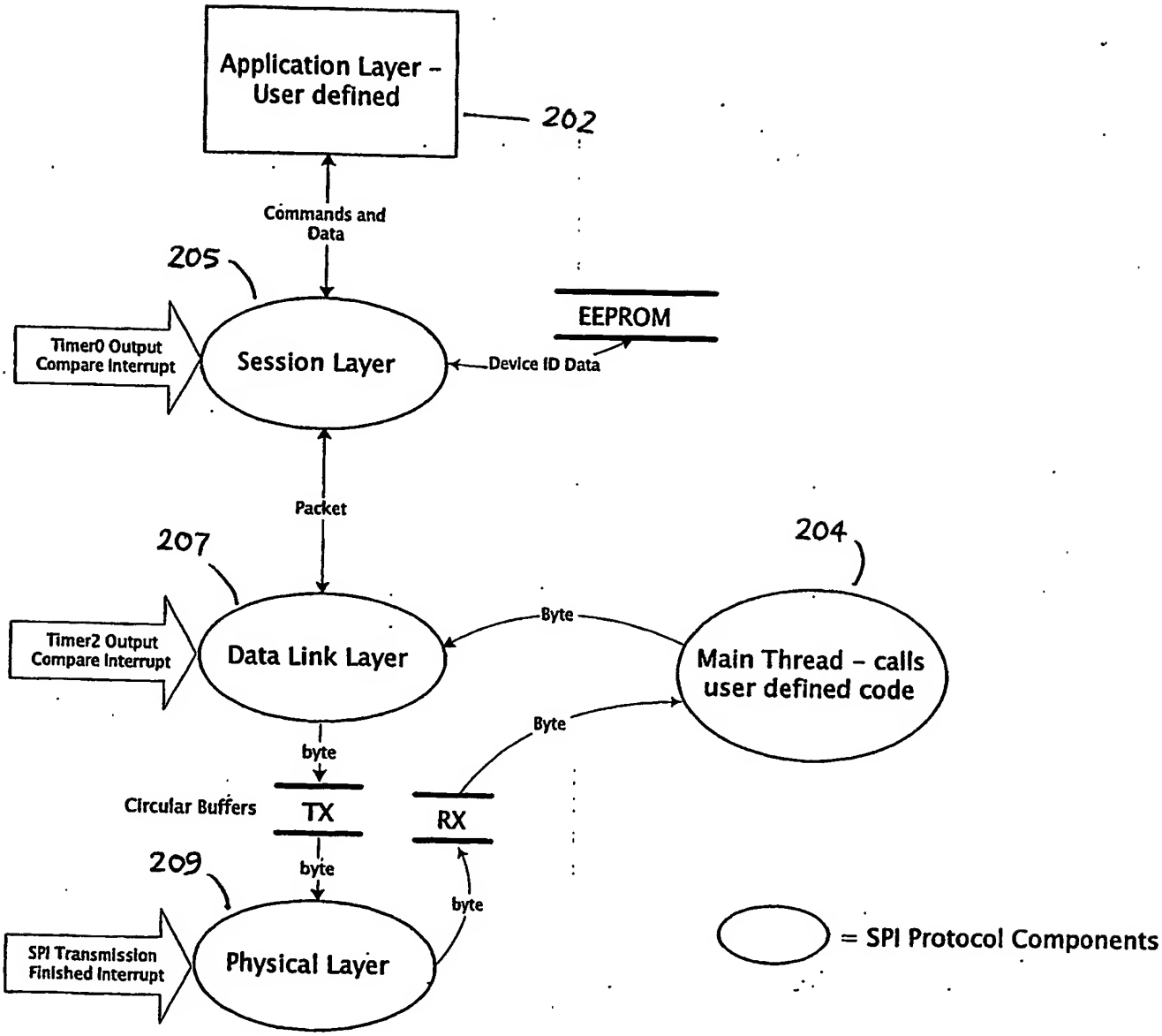
Application Layer - User defined
202
Commands and Data
205
EEPROM
Timer0 Output Compare Interrupt
Session Layer
Device ID Data
Packet
207
204
Timer2 Output Compare Interrupt
Data Link Layer
Byte
Main Thread - calls user defined code
Byte
byte
Circular Buffers
TX
RX
209
byte
byte
SPI Transmission Finished Interrupt
Physical Layer
= SPI Protocol Components

FIG. 21

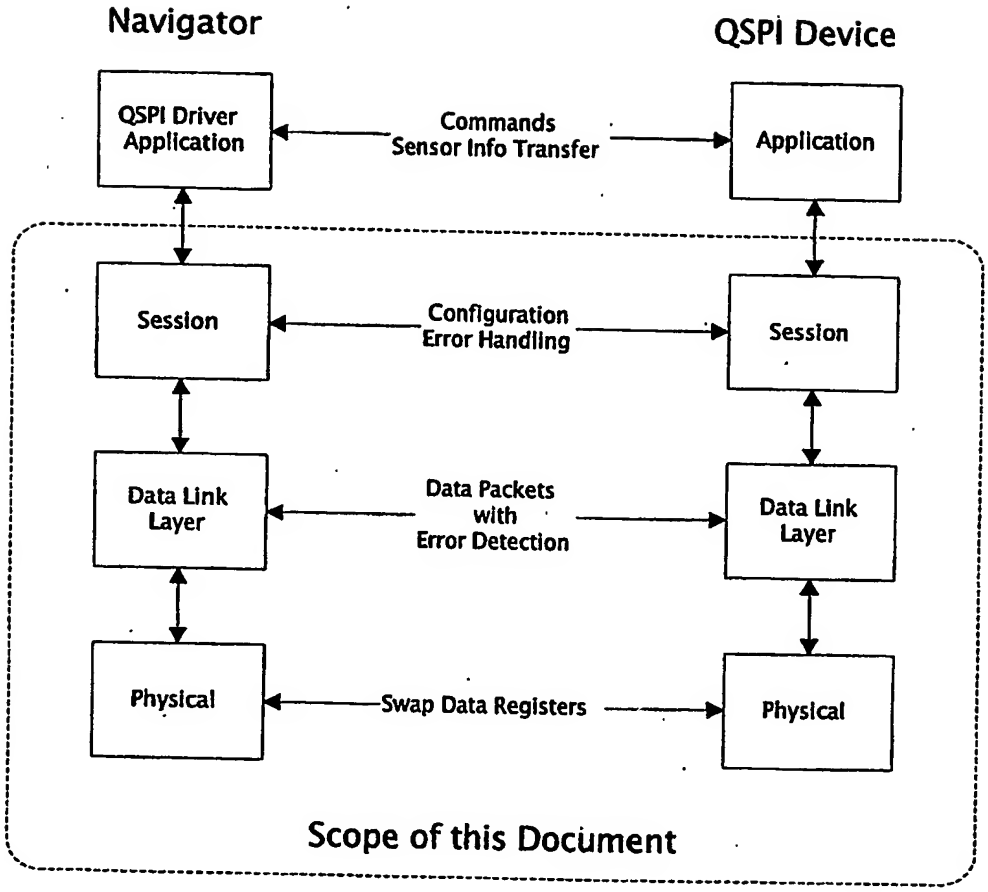
Navigator
QSPI Device
QSPI Driver Application
Commands
Sensor Info Transfer
Application
Session
Configuration
Error Handling
Session
Data Link Layer
Data Packets
with
Error Detection
Data Link Layer
Physical
Swap Data Registers
Physical
Scope of this Document

FIG. 22

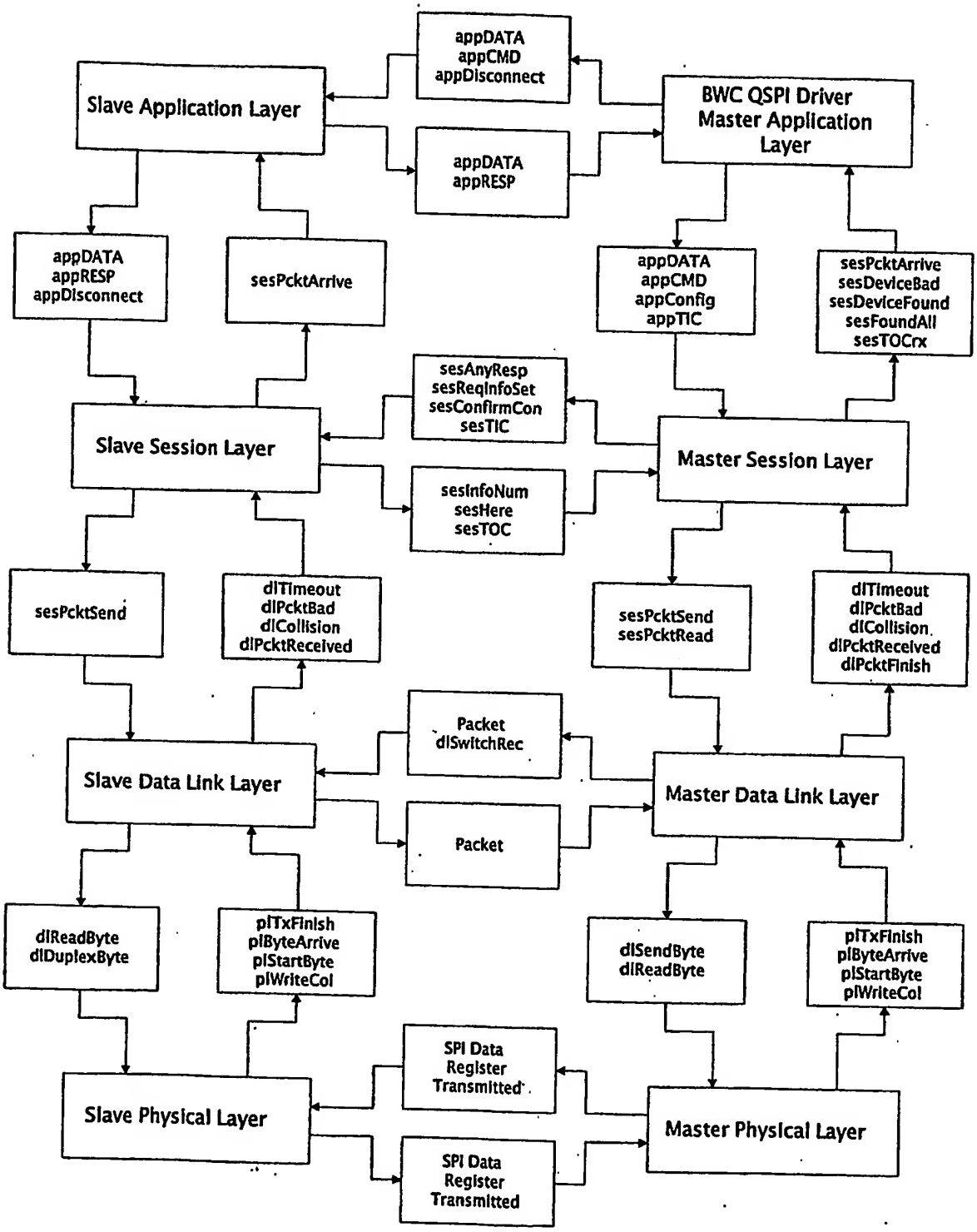
appDATA
appCMD
appDisconnect
Slave Application Layer
BWC QSPI Driver
Master Application
Layer
appDATA
appRESP
appDATA
appRESP
appDisconnect
sesPcktArrive
appDATA
appCMD
appConfig
appTIC
sesPcktArrive
sesDeviceBad
sesDeviceFound
sesFoundAll
sesTOCrx
sesAnyResp
sesReqInfoSet
sesConfirmCon
sesTIC
Slave Session Layer
Master Session Layer
sesInfoNum
sesHere
sesTOC
sesPcktSend
dlTimeout
dlPcktBad
dlCollision
dlPcktReceived
sesPcktSend
sesPcktRead
dlTimeout
dlPcktBad
dlCollision
dlPcktReceived
dlPcktFinish
Packet
dlSwitchRec
Slave Data Link Layer
Master Data Link Layer
Packet
dlReadByte
dlDuplexByte
plTxFinish
plByteArrive
plStartByte
plWriteCol
dlSendByte
dlReadByte
plTxFinish
plByteArrive
plStartByte
plWriteCol
SPI Data
Register
Transmitted
Slave Physical Layer
Master Physical Layer
SPI Data
Register
Transmitted

FIG. 23

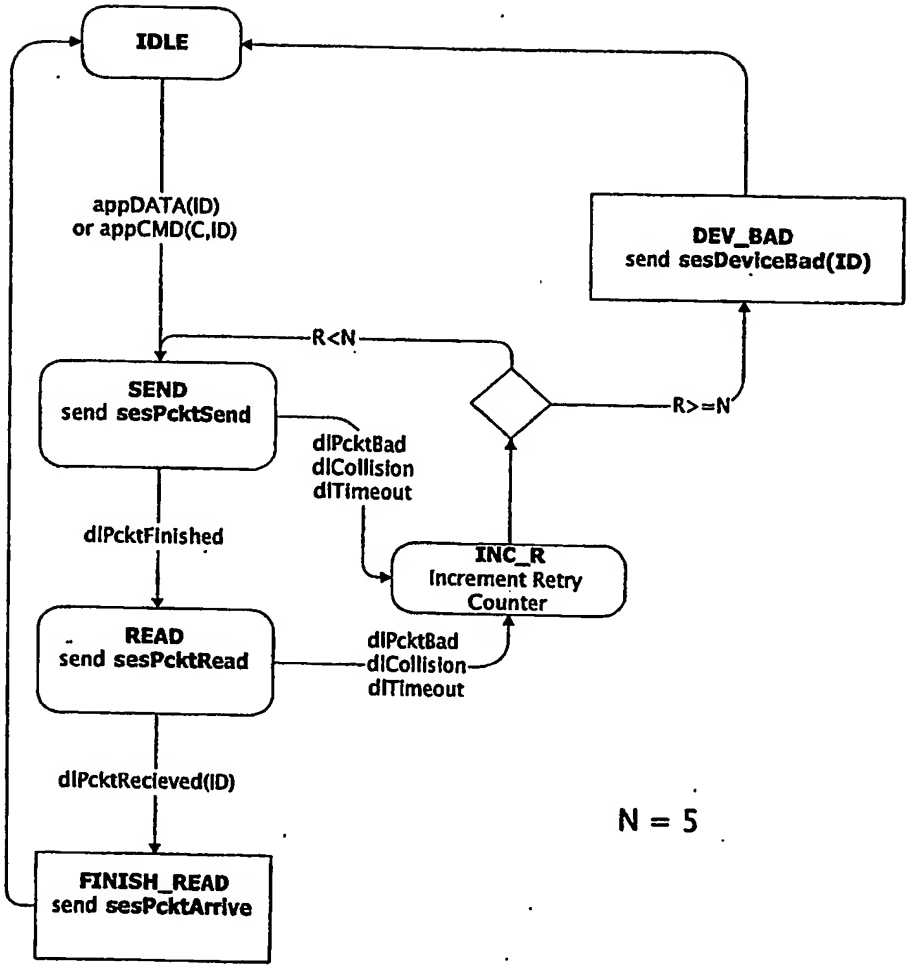
IDLE
appDATA(ID)
or appCMD(C,ID)
DEV_BAD
send sesDeviceBad(ID)
R<N
SEND
send sesPcktSend
R>=N
dlPcktBad
dlCollision
dlTimeout
dlPcktFinished
INC_R
Increment Retry
Counter
READ
send sesPcktRead
dlPcktBad
dlCollision
dlTimeout
dlPcktRecieved(ID)
N = 5
FINISH_READ
send sesPcktArrive

FIG. 24

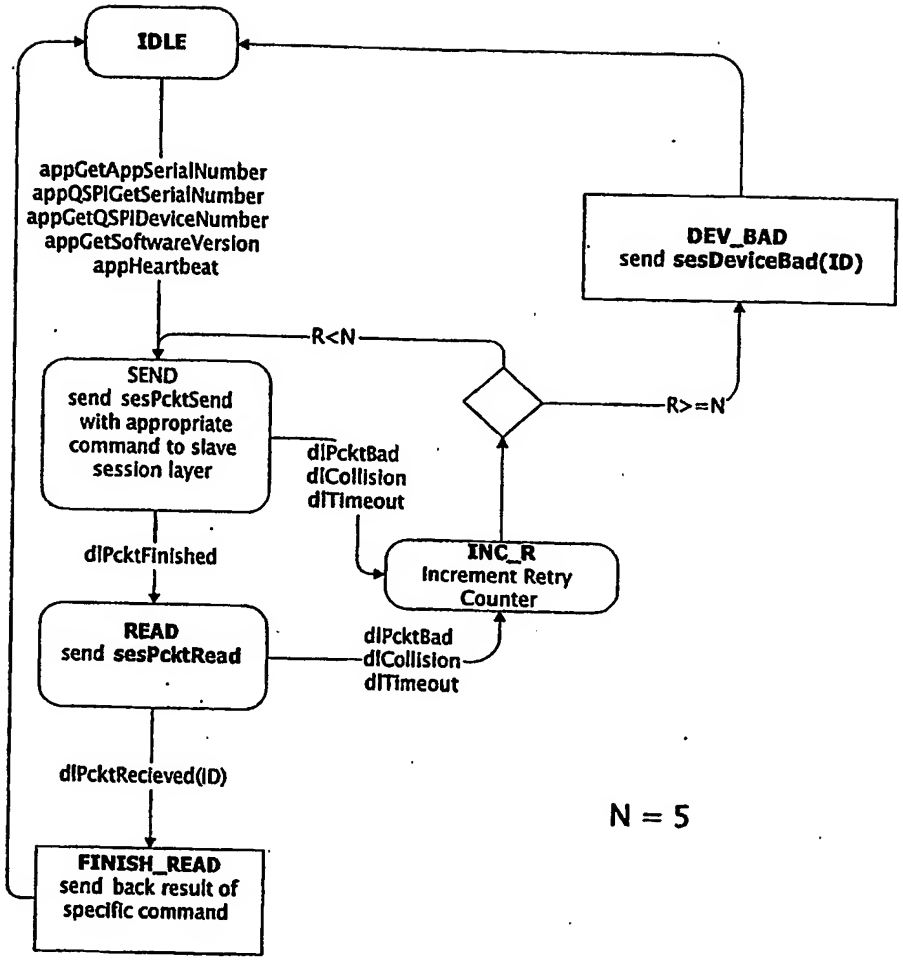
IDLE
appGetAppSerialNumber
appQSPIGetSerialNumber
appGetQSPIDeviceNumber
appGetSoftwareVersion
appHeartbeat
DEV_BAD
send sesDeviceBad(ID)
R<N
SEND
send sesPcktSend
with appropriate
command to slave
session layer
R>=N
dlPcktBad
dlCollision
dlTimeout
dlPcktFinished
INC_R
Increment Retry
Counter
READ
send sesPcktRead
dlPcktBad
dlCollision
dlTimeout
dlPcktRecieved(ID)
N = 5
FINISH_READ
send back result of
specific command

FIG. 25

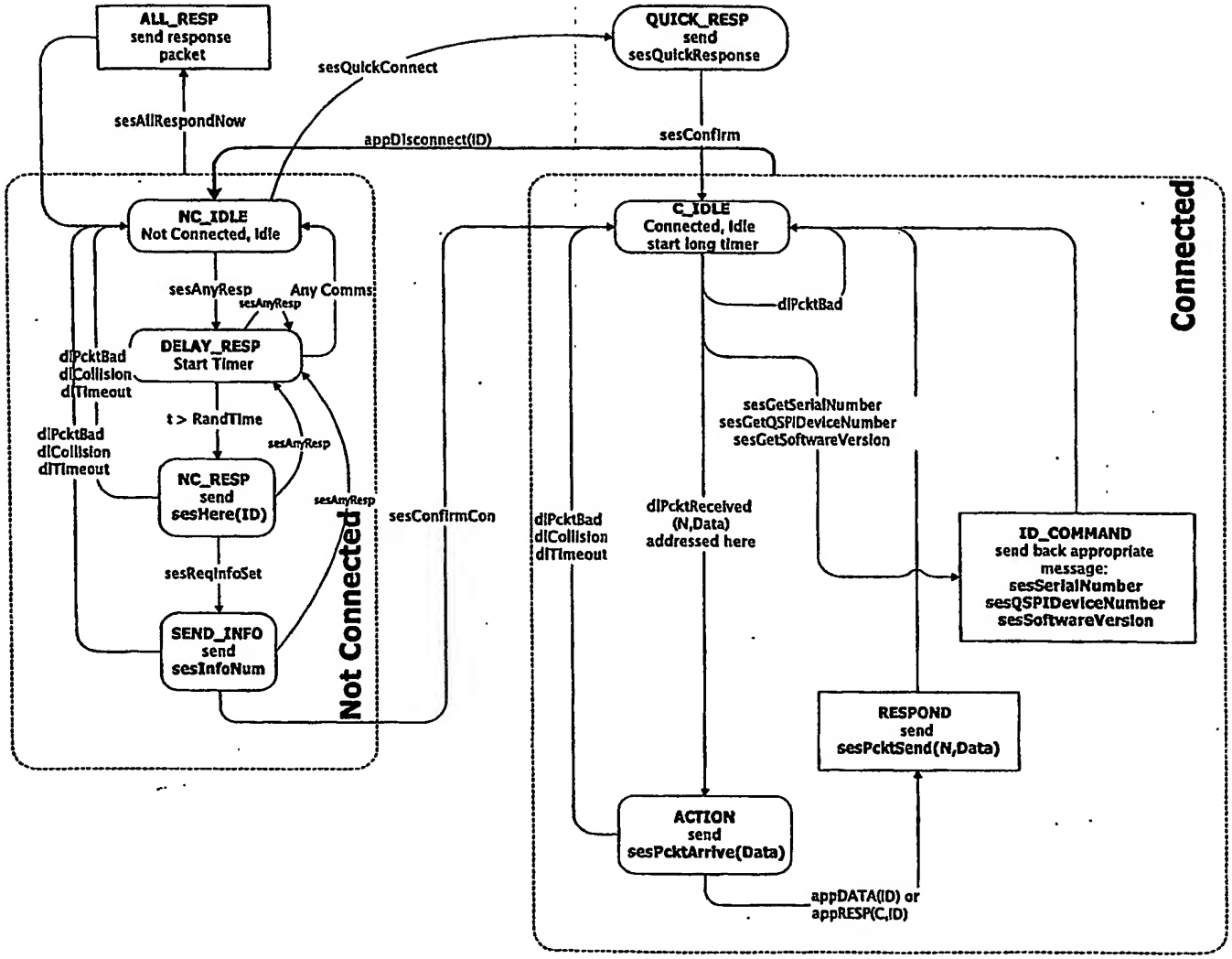
ALL_RESP
send response packet
QUICK_RESP
send
sesQuickResponse
sesQuickConnect
sesAllRespondNow
appDisconnect(ID)
sesConfirm
NC_IDLE
Not Connected, Idle
C_IDLE
Connected, Idle
start long timer
Connected
sesAnyResp
Any Comms
sesAnyResp
DELAY_RESP
Start Timer
dlPcktBad
dlCollision
dlTimeout
dlPcktBad
t > RandTime
sesGetSerialNumber
sesGetQSPIDeviceNumber
sesGetSoftwareVersion
dlPcktBad
dlCollision
dlTimeout
sesAnyResp
NC_RESP
send
sesHere(ID)
sesAnyResp
sesConfirmCon
dlPcktBad
dlCollision
dlTimeout
dlPcktReceived
(N,Data)
addressed here
ID_COMMAND
send back appropriate message:
sesSerialNumber
sesQSPIDeviceNumber
sesSoftwareVersion
sesReqInfoSet
SEND_INFO
send
sesInfoNum
Not Connected
RESPOND
send
sesPcktSend(N,Data)
ACTION
send
sesPcktArrive(Data)
appDATA(ID) or
appRESP(C,ID)

FIG. 26

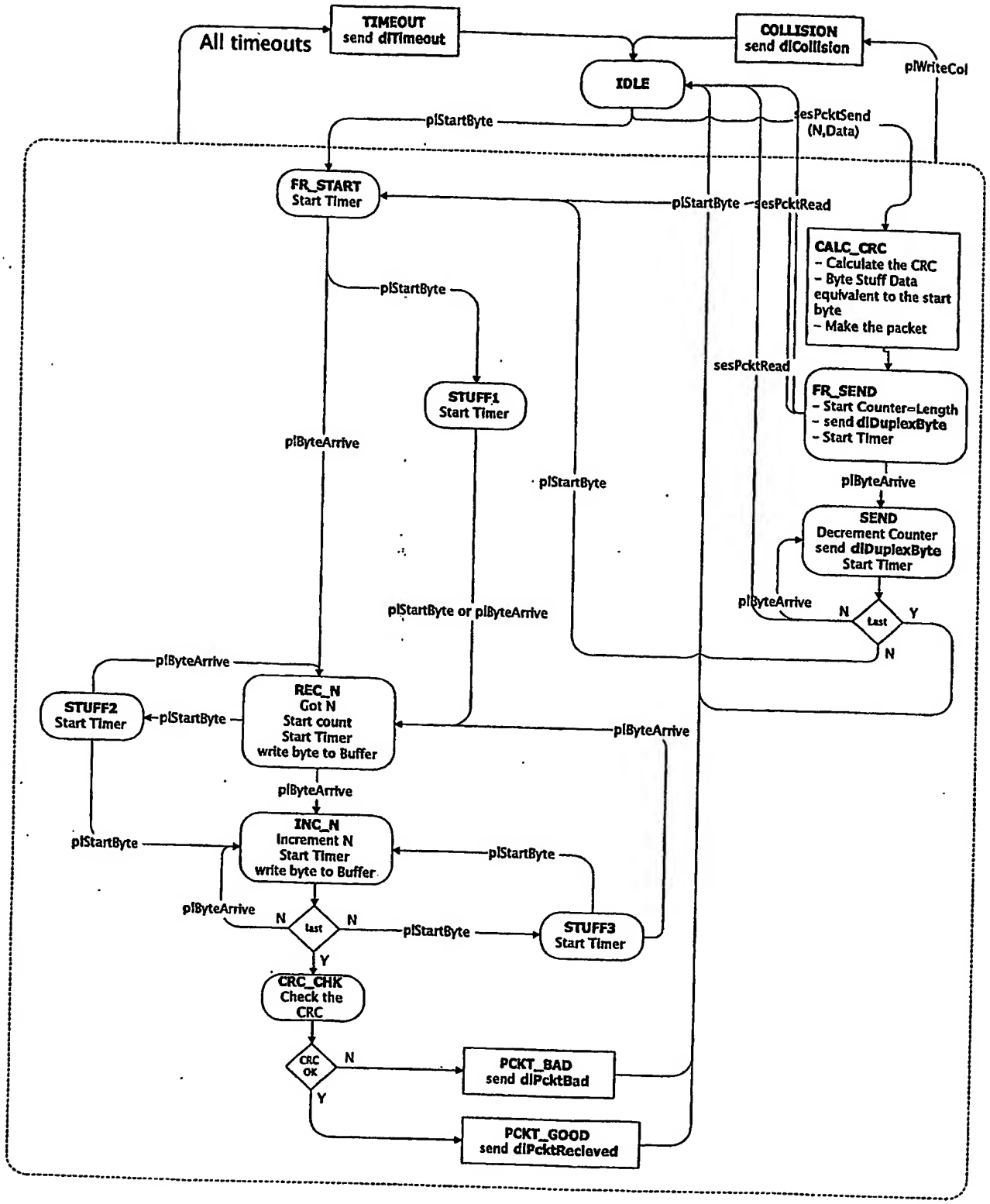
All timeouts
TIMEOUT
send dlTimeout
COLLISION
send dlCollision
plWriteCol
IDLE
plStartByte
sesPcktSend
(N,Data)
FR_START
Start Timer
plStartByte
sesPcktRead
CALC_CRC
- Calculate the CRC
- Byte Stuff Data equivalent to the start byte
- Make the packet
plStartByte
STUFF1
Start Timer
sesPcktRead
FR_SEND
- Start Counter=Length
- send dlDuplexByte
- Start Timer
plByteArrive
plStartByte
plByteArrive
SEND
Decrement Counter
send dlDuplexByte
Start Timer
plStartByte or plByteArrive
plByteArrive
N
Last
Y
N
plByteArrive
STUFF2
Start Timer
plStartByte
REC_N
Got N
Start count
Start Timer
write byte to Buffer
plByteArrive
plByteArrive
plStartByte
INC_N
Increment N
Start Timer
write byte to Buffer
plStartByte
plByteArrive
N
Last
N
plStartByte
STUFF3
Start Timer
Y
CRC_CHK
Check the CRC
CRC OK
N
PCKT_BAD
send dlPcktBad
Y
PCKT_GOOD
send dlPcktRecieved

FIG. 27

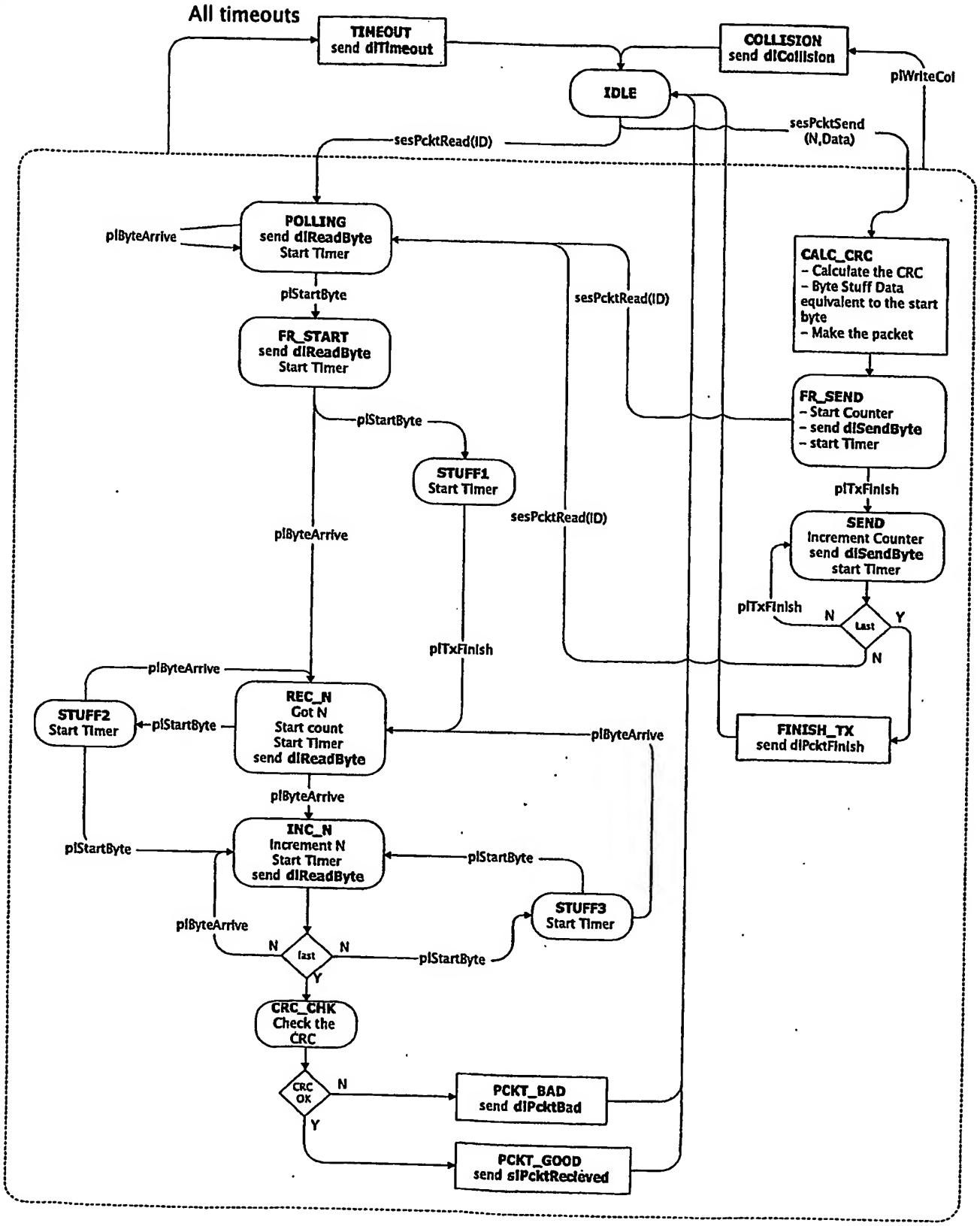
All timeouts
TIMEOUT
send dlTimeout
COLLISION
send dlCollision
plWriteCol
IDLE
sesPcktSend
(N,Data)
sesPcktRead(ID)
POLLING
send dlReadByte
Start Timer
plByteArrive
plStartByte
FR_START
send dlReadByte
Start Timer
sesPcktRead(ID)
CALC_CRC
- Calculate the CRC
- Byte Stuff Data equivalent to the start byte
- Make the packet
FR_SEND
- Start Counter
- send dlSendByte
- start Timer
plTxFinish
SEND
Increment Counter
send dlSendByte
start Timer
plTxFinish
N
Last
Y
N
STUFF1
Start Timer
sesPcktRead(ID)
plByteArrive
plTxFinish
plByteArrive
STUFF2
Start Timer
plStartByte
REC_N
Got N
Start count
Start Timer
send dlReadByte
plByteArrive
FINISH_TX
send dlPcktFinish
plByteArrive
INC_N
Increment N
Start Timer
send dlReadByte
plStartByte
plStartByte
STUFF3
Start Timer
plByteArrive
N
last
N
Y
plStartByte
CRC_CHK
Check the CRC
CRC OK
N
Y
PCKT_BAD
send dlPcktBad
PCKT_GOOD
send slPcktRecieved

FIG. 28